TENTAMEN COMPILERCONSTRUCTIE
Maandag 19 december 2016, 14:00 – 17:00 uur

Dit tentamen bestaat uit vijf opgaven, waarbij steeds tussen [en] staat hoeveel punten er ongeveer mee te verdienen zijn. In totaal zijn er 100 punten te verdienen.

Als je het antwoord op een onderdeel niet weet, en je hebt dat antwoord nodig bij een later onderdeel, dan kun je het antwoord 'kopen' bij de docent.

Als er bij een opgave gevraagd wordt om uitleg of motivatie van je antwoord, is het belangrijk dat je die ook geeft.

---

1. [9 pt] Noem drie soorten informatie over een variabele die in de *symbol table* worden bijgehouden. In welke fase van het compileerproces wordt elk van deze soorten informatie in de symbol table gezet? Motiveer je antwoorden.

---

2. [24 pt]

   (a) Laat α een string van symbolen uit een context-vrije grammatica $G = (V, \Sigma, S, P)$ zijn, en laat $A \in V$ een variabele zijn. Leg duidelijk (en volledig) uit wat FIRST(α) en FOLLOW(A) zijn. Wat zit er in?

   Beschouw de context-vrije grammatica G met startvariabele S en de volgende producties:

$$\begin{aligned} S &\rightarrow AaB \mid bA \\ A &\rightarrow Bb \mid \epsilon \\ B &\rightarrow aS \mid cB \end{aligned}$$

   De terminalen in G zijn dus a, b, c.

   (b) Bepaal voor elke variabele in de grammatica G zowel de FIRST- als de FOLLOW-verzameling.

   (c) Construeer de top-down *parsing table* bij grammatica G. Wellicht ten overvloede: dit is dus iets anders dan de SLR parsing table.

   (d) Is de grammatica G een LL(1) grammatica? Motiveer je antwoord.

---

3. [19 pt] In plaats van een syntax tree kunnen we ook een gerichte acyclische graaf (*Directed Acyclic Graph = DAG*) gebruiken om de structuur van een programma weer te geven.

   (a) Noem twee voordelen van een gerichte acyclische graaf ten opzichte van een syntax tree.

   (b) Teken zowel een rechttoe-rechtaan syntax tree als een gerichte acyclische graaf voor de expressie $a + b + c * (a + b) + c$.

   (c) Behalve enkele expressies kunnen ook reeksen van instructies in een *basic block* met een gerichte acyclische graaf worden weergegeven. Teken de gerichte acyclische graaf bij de volgende reeks instructies:

```
d=a+b
e=d+c
b=a+b
c=a+b
```

   Maak duidelijk welke informatie je allemaal in/bij de knopen in de graaf opslaat.

   (d) Leg uit hoe je informatie over welke variabelen live zijn aan het eind van een basic block kunt combineren met de gerichte acyclische graaf van dat block, om dead code te elimineren.

---

4. [27 pt] Tijdens het genereren van drie-adres code voor boolese expressies en *flow-of-control* instructies weten we bij goto-instructies vaak niet onmiddellijk waar we naartoe moeten springen. We kunnen *backpatching* gebruiken om dit achteraf op te lossen. Hierbij krijgt de variabele B in de grammatica (overeenkomend met een boolese expressie) attributen *truelist* en *falselist*. De variabele S (overeenkomend met een instructie) krijgt een attribuut *nextlist*.

   (a) Leg voor elk van deze drie lijsten uit wat de lijst bevat.

   Naast deze variabelen gebruiken we hieronder een hulpvariabele M (met een attribuut *instr*) en een variabele E (overeenkomend met een rekenkundige expressie, met een attribuut *addr*).

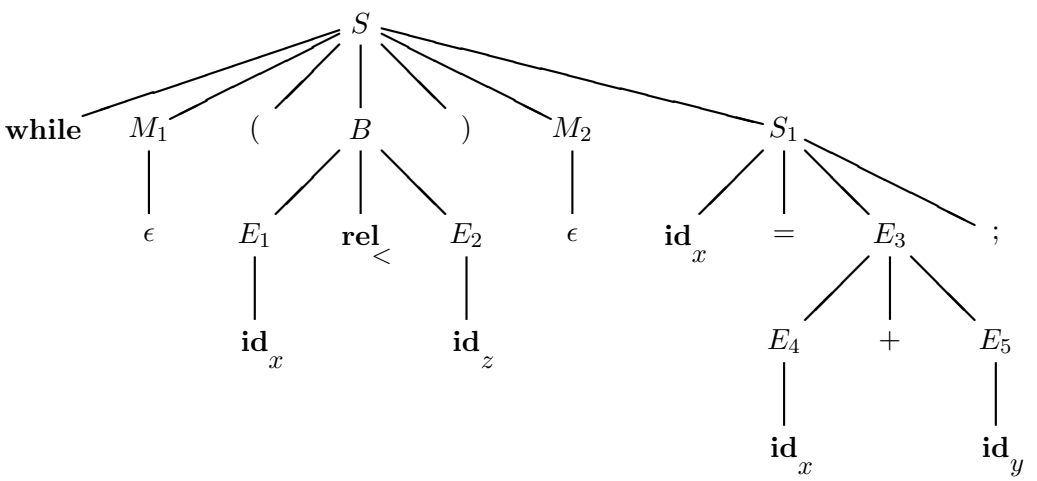
   Beschouw het volgende *translation scheme* voor het genereren van de genoemde drie-adres code:

$S \rightarrow \textbf{while}\ M_1\ (B)\ M_2\ S_1$	{	$backpatch(S_1.nextlist, M_1.instr);$ $backpatch(B.truelist, M_2.instr);$ $S.nextlist = B.falselist;$ $gen('\texttt{goto}\ M_1.instr');$ }
$S \rightarrow \textbf{id} = E;$	{	$S.nextlist = \textbf{null};$ $gen(top.get(\textbf{id}.lexeme)\ ' =' \ E.addr);$ }
$E \rightarrow E_1 + E_2$	{	$E.addr = \textbf{new}\ Temp();$ $gen(E.addr\ ' =' \ E_1.addr\ ' +' \ E_2.addr);$ }
$E \rightarrow \textbf{id}$	{	$E.addr = top.get(\textbf{id}.lexeme);$ }
$B \rightarrow E_1\ \textbf{rel}\ E_2$	{	$B.truelist = makelist(nextinstr);$ $B.falselist = makelist(nextinstr + 1);$ $gen('\texttt{if}'\ E_1.addr\ \textbf{rel}.op\ E_2.addr\ '\texttt{goto}\ \_');$ $gen('\texttt{goto}\ \_');$ }
$M \rightarrow \epsilon$	{	$M.instr = nextinstr;$ }

   De afleidingsboom (*parse tree*) bij bovenstaande grammatica (met startvariabele S) voor het volgende 'programma':

```
while (x<z)
  x=x+y;
```

   ziet er als volgt uit:



(b) Pas bij bovenstaande afleidingsboom de semantische acties toe zoals beschreven in het translation scheme. Geef bij elke variabele in de boom aan wat de attributen (*truelist*, *falselist*, *nextlist*, *instr* en/of *addr*) worden. Geef ook de resulterende drie-adres code.

Ga ervanuit dat deze drie-adres code begint op instructienummer 100, en dat benodigde nieuwe tijdelijke variabelen achtereenvolgens t1, t2, ... heten.

(c) Leg uit wat er volgens het translation scheme gebeurt (de semantische actie) bij de productie

$$S \rightarrow \textbf{while}\ M_1\ (B)\ M_2\ S_1$$

Leg ook uit waarom dat gebeurt.

N.B.: het gaat er hier om wat er in het algemeen gebeurt bij deze productie, en niet zozeer wat er gebeurt in het geval van ons voorbeeld'programma'.

---

5. [21 pt]

(a) Om op lokaal niveau efficiënt registers te alloceren, kunnen we gebruik maken van *register descriptors* en *address descriptors*. Wat verstaan we onder deze twee 'descriptors'?

(b) We willen nu de volgende drie-adres code omzetten in assembly:

```
d=a+b
e=d+c
a=e
c=a+b
a=b-c
```

Gegeven is dat alle variabelen live-on-exit zijn.

Zet de gegeven drie-adres code instructie-voor-instructie om in assembly, waarbij je efficiënt met registers omgaat. Je mag hierbij gebruik maken van

- drie registers R1, R2 en R3, die aanvankelijk leeg zijn
- assembly instructies van de volgende vorm:

```
ADD Ri, Rj, Rk
SUB Ri, Rj, Rk
LD Ri, x
ST x, Ri
```

Hierbij zijn Ri, Rj en Rk registers (eventueel dezelfde), en is x een variabele. Bij ADD en SUB komt het resultaat in Ri te staan.

Aan het eind hoef je de inhoud van de registers niet met allemaal store instructies veilig op te slaan.

Geef, behalve de resulterende assembly, in een overzichtelijke tabel de inhoud van de register descriptors en de address descriptors

- aan het begin
- en na ieder stukje assembly dat met een drie-adres instructie overeenkomt.

(dus in totaal zes keer).

Geef bij iedere drie-adres instructie een korte motivatie van de keuze voor de gebruikte registers.
