Error checking.

Voornamelijk is vandaag aandacht geschonken aan de wijze van terugmelding.

De vertaler nummert de niet-lege regels van de programmatext. Als bijproduct geeft hij via de output van een aantal labels het regelnummer van de regel, waarin ze voorkomen. Procedure declarations zijn ook mooi.
Opm.1 De vertaler zal hier identifiers in 1 alphabet uitprinten.
Opm.2. De output kan aangeven of het een label dan wel een procedure identifier is; kan zelfs iets van de blokstructuur proberen aan te geven. Als de text zo slechts, dat we een lower bound voor label aanzien, dan zal deze print wel wat cryptisch worden!

Probleem: in de dynamische plaatsbepaling ontmoeten we ook MCP's. In het minimale geval zeggen we, dat elke bibliotheekprocedure op 1 regel staat. Een alternatief is, om elke MCP moedwillig in een aantal regels onder te verdelen; netjes is om bij de terugmelding ook de naam van de MCP te vermelden. (Dit kon er wel eens toe leiden, dat we elke MCP een naam geven.)

Statische checking.

De vertaling zal bestaan uit een aantal passes (eigenlijk hoop ik van niet te veel). We zullen proberen om bijde detectie van een fout althans de pass, waarin deze fout gedetecteerd wordt, af te maken met de bedoeling meer fouten van het zelfde allooi te vinden. (Of dat altijd kan, is heel erg de vraag: is niet de executie "de laatste pass"?)

Wat betreft error checking zie ik maar 2 passes noodzakelijk optreden:
1) analyse van delimiter structure en opbouw van de naamlijst
2) type checking etc, dwz. alles, waarvoor je de declaratie gezien moet hebben.

Opm.3. Als ik in de pass, waarin de regelnummers uitgedeeld worden ook al check, zullen die foutmeldingen er wel doorheen komen; dit hindert niet.
Opm.4. Een voordeel van Cor's regelnummering is, dat deze uniek is, ook in die gevallen, waarin de systactische structuur macroscopisch niet klopt.!

Ik stel me voor, dat de vertaler in een left to right pass, waarin de regelscheidingen in de source text nog herkenbaar zijn, het uiteindelijke programma definitief over de segmenten verdeelt. Op dat moment kan de vertaler dus een lijst maken van invariante beginadressen (programma adressen), in de volgorde van opklimmend regelnummer. (Ik neem aan, dat voor de opeenvolgende programma segmenten opeenvolgende SV's gebruikt kunnen worden, zodat we die niet hoeven te ketenen.) Bij de dynamische plaatsmelding kan deze lijst geraadpleegd worden

Statische errormelding zal bestaan uit:
1) korte beschrijving van de gemaakte fout. (Omdat we een trommel en een high speed printer hebben, moeten we hiermee niet te karig zijn, dus niet "fout 7")
2) regelnummer.

Dynamische errormedling zal bestaan uit:
1) korte beschrijving van de soort fout (zie boven)
2)plaats.

Ruwweg komt deze opgave op het volgende neer:
je geeft een regelnummer (afgeleid uit een invariant adres). Via inspectie van de stapel vindt je de aard van het niveau (te weten binnenblok, parameter of fictitious block). Deze aard meldt je; als het een binnenblok is, ga je de dynamische ketting af, totdat je een fictitious block vindt (of de buitenkant). Dan heb je in de stapel een returnadres

en vervolgens ga je dat analyseren. Aangezien parameters geen binnenblokken hebben zijn er dus drie mogelijkheden:
1) parametersituatie
2) block (dwz, buitenste of binnenblok van een procedure)
3) fictitious block.

Opm. Als we het fictiotious block gevonden hebben, staat in de stapel het invariante starting address van de procedure, die we wilden verlaten. Een lijstje is voldoende om de procedure identifier er bij te verschaffen.

Post mortem.

Als we ons op het standpunt stellen, dat een uniforme post mortem reactie (bv. altijd alles) irreeel is, dan laat zich dit als volgt sturen. Vooraan de band (program heading) is een entry, die de aard van de post mortem dump aangeeft. De hier mogelijke notities worden met 1 uitgebreid, die betekent "als een dynamische fout optreedt, vraag dan via toetsenbord". Dit geeft alle mogelijke flexibiliteit zonder last als je er geen behoefte aan hebt. (Je kunt dit uitbreiden met bv. "De lege post mortem notitie in de heading betekent niets".)

Verder hebben we te dumpen anoniemen, scalairen en arrays.
ad1) de anoniemen zijn wat moeilijk, omdat we de interpretatie niet in de stapel vinden. De dumper kan integers uitprinten en zo mogelijk (tekenconsistentie) ten overvloede paren ook als drijvend getal. De interpretatie van deze rommel is in elk geval voorbehouden aan specialisten (men denke bv. aan de verwisseling van primaries!)

ad2) de scalairen hebben twee moelijkheden
2a) bij de scalairen van een expliciet programma moet je, als je enig fatsoen hebt, de identifiers -zij het dan gereduceerd tot een alphabet- er bij fokken Taakje voor de vertaler en de trommel!
2b) bij de handgecodeerde MCP's heb je niet een twee drie een naamlijst. Het alternatief dat we voor de anonymen bedacht hebben, kon hier wel eens aangewezen zijn (format kan hier nl. wisselen.)

ad3) arrays kunnen zo groot zijn. Wat we met arrays in MCP's moeten doen is nog minder duidelijk, maar op dit ogenblik willen we daar niet te veel over denken.

Statische checks zullen omvatten: delimiter structure, identifier matching, correct gebruik (subscripts etc) en type checking. Identifier matching omvat een duidige correspondentie vaststellen tussen gebruik en declaratie, vangen van meervoudige declaratie in een blok, multipele formals, complete specificatie, controle van de value list etc.

Dynamische checks omvatten: actual formal correspondence (inclusief tellen aantal parameters) array declaratie (lower bound niet groter dan de upper bound) subscript values within bounds, formal array's test op aantal subscripts, formal left hand side consistency by multiple assignment, complex → real barrier, sqrt, ln, to the power integer division, input output control, integer overflow

Wat we niet checken is: delen door 0, feitelijke assignment in type procedure, undefined variable (grote voorkeur, althans in het begin, voor de uniforme reactie) niet eindige recursie (maar die wordt via de stack control gevangen) en de blinde loop