TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

AutoLISP voor beginners : een beknopte handleiding

Citation for published version (APA):
Marinissen, E. J., & Deckers, R. T. C. (1991). *AutoLISP voor beginners : een beknopte handleiding*. Technische Universiteit Eindhoven.

Document status and date:
Gepubliceerd: 01/01/1991

Document Version:
Uitgevers PDF, ook bekend als Version of Record

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 04. Oct. 2023

TECHNISCHE UNIVERSITEIT EINDHOVEN

Fakulteit Wiskunde & Informatika

AutoLISP VOOR BEGINNERS

een beknopte handleiding

Erik Jan Marinissen Robert Deckers

in opdracht van
Fakulteit Werktuigbouwkunde
Vakgroep WOC - Sektie CAE

oktober 1991

AutoLISP VOOR BEGINNERS

een beknopte handleiding

Erik Jan Marinissen[1] Robert Deckers

oktober 1991

Technische Universiteit Eindhoven,
Fakulteit Wiskunde & Informatika,
Postbus 513, 5600 MB EINDHOVEN
e-mail: wsinem@win.tue.nl

Samenvatting

AUTOLISP is de programmeertaal van het populaire CAD-pakket AUTOCAD. AUTOLISP biedt de gevorderde AUTOCAD gebruiker de mogelijkheid zijn CAD-pakket helemaal aan te passen aan zijn specifieke wensen. Met AUTOLISP wordt het mogelijk zelf nieuwe AUTOCAD kommando's te definiëren of bestaande kommando's te herdefiniëren. Achter deze kommando's kunnen algoritmen en numerieke berekeningen schuilen, die in AUTOLISP geprogrammeerd worden.

Dit dokument geeft een introduktie tot het gebruik van AUTOLISP. Na korte inleidingen over AUTOCAD en de programmeertaal LISP, worden de basisfunkties van AUTOLISP uitgebreid behandeld. De theorie wordt rijkelijk geïllustreerd met voorbeelden. Hierna wordt een voorbeeld gegeven van een realistisch AUTOLISP programma. Tenslotte wordt ingegaan op standaard bij AUTOCAD bijgeleverde AUTOLISP programma's en problemen die een aankomend AUTOLISP programmeur kan tegenkomen.

[1]Ook: Philips Research Laboratories, Postbus 80.000, 5600 JA EINDHOVEN,
e-mail: marinis@prl.philips.nl

Inhoud

1 Inleiding

AUTOCAD is een *Computer Aided Design* (CAD) pakket met een open struktuur. Dit houdt in dat het zelf een algemeen bruikbaar tekenpakket is, maar dat het de gebruiker mogelijkheden biedt om het voor zijn eigen specifieke toepassingen aan te passen. Het schrijven van *makro's* is hiervoor één mogelijkheid. De meest geavanceerde mogelijkheid die AUTOCAD daarvoor biedt, is echter het gebruik van AUTOLISP. AUTOLISP is een op LISP gebaseerde programmeertaal, die het mogelijk maakt zelf eigen AUTOCAD kommando's te definiëren of bestaande kommando's te herdefiniëren. Achter deze nieuwe kommando's gaan (al dan niet gekompliceerde) algoritmen schuil, die in AUTOLISP geprogrammeerd worden.

Deze handleiding behandelt AUTOLISP voor AUTOCAD gebruikers die nog geen ervaring hebben met AUTOLISP. De hoofdstukken 1 tot en met 4 hebben een algemeen, inleidend karakter. In Hoofdstuk 1 wordt ingegaan op de veronderstelde voorkennis van de lezer en worden verder enkele notationele afspraken gemaakt. Hoofdstuk 2 behandeld de hardware en software die nodig is voor het gebruik van AUTOCAD en AUTOLISP. Hoofdstuk 3 geeft een korte introduktie op AUTOCAD. De lezer wordt bekend verondersteld met AUTOCAD; in dit hoofdstuk worden echter de aspekten van AUTOCAD die belangrijk zijn voor het gebruik van AUTOLISP nog even opgesomd. Hoofdstuk 4 gaat in op de geschiedenis en eigenschappen van LISP als algemene programmeertaal en de relatie tussen LISP en AUTOLISP. Hoofdstuk 5 behandelt alle basisfunkties van AUTOLISP. Het gebruik daarvan wordt steeds aan de hand van kleine voorbeeldjes toegelicht. Hoofdstuk 6 geeft een realistisch voorbeeld van het gebruik van AUTOLISP. Hierin komen veel van de in Hoofdstuk 5 behandelde konstrukties in terug. Hoofdstuk 7 vertelt hoe men de standaard bijgeleverde AUTOLISP programma's binnen je bereik kunt krijgen. Omdat het pad van een programmeur (en zeker van een beginnende programmeur) niet (altijd) over rozen gaat, is Hoofdstuk 8 gewijd aan problemen die kunnen optreden en hoe deze op te lossen. Hoofdstuk 9 besluit deze handleiding. Achteraan dit dokument zijn opgenomen een lijst met literatuurverwijzingen en een appendix, met daar een opsomming van de meest gebruikte standaard funkties van AUTOLISP.

1.1 Voorkennis

Om dit dokument goed te kunnen lezen en begrijpen, is voorkennis nodig van eenvoudig gebruik van:

- de komputer (bijvoorbeeld een personal computer) waarop en het operating systeem (bijvoorbeeld MS-DOS) waaronder AUTOCAD werkt;
- de tekst editor die aanwezig is op de komputer;

- AUTOCAD.

Programmeerervaring (wellicht in andere omgevingen en in andere programmeertalen) kan voordelig zijn, maar is niet vereist, evenmin als kennis van LISP of AUTOLISP.

1.2 Notationele konventies

In deze handleiding worden regelmatig voorbeelden gegeven van kommando's en programma's, kortom, van interaktie met AUTOCAD. Letterlijke komputer in- en uitvoer wordt altijd weergegeven in het zgn. `typewriter font`. Om invoer van uitvoer te scheiden wordt invoer die door de gebruiker moet worden opgegeven onderlijnd.

2 Konfiguratie voor het gebruik van AutoLISP

AUTOLISP werkt op alle systemen waar AUTOCAD is geïnstalleerd. Dit kunnen UNIX, DOS, OS/2, AEGIS, VMS en Macintosh systemen zijn. Deze handleiding is gebaseerd op AUTOCAD en AUTOLISP gebruik op personal computers. Vrijwel alles gaat echter ook op voor andere komputers en andere operating systems. De lezer van deze handleiding wordt verondersteld kennis te hebben van de komputer en het operating system waar hij mee werkt.

AUTOLISP programma's zijn ASCII files die door AUTOCAD ingelezen en uitgevoerd kunnen worden. Echter, AUTOCAD zelf heeft geen tekst editor om deze files te produceren. Daarom is het onontbeerlijk om over een tekst editor te kunnen beschikken die teksten als ASCII files kan op slaan. Hiervoor kan de standaard PC editor `edlin` gebruikt worden. Veel gebruikersvriendelijker en in PC omgevingen veelvuldig aanwezig is de Norton Editor, maar ook editors van andere programmeertalen (Turbo Pascal) of tekstverwerkers (WordPerfect) kunnen worden gebruikt. In het laatste geval moet er op gelet worden dat files worden bewaard als ASCII files, dus zonder de kontrole karakters van de tekstverwerker.

3 AutoCAD

AUTOCAD is één van de meest verkochte Computer Aided Design (CAD) pakketten ter wereld. [Ot 89] schrijft dat in 1988 meer dan 130.000 pakketten verkocht waren. AUTOCAD is bedoeld voor twee-dimensionale tekeningen, maar heeft ook mogelijkheden om drie-dimensionaal te werken.

In het hele skala van CAD systemen dat tegenwoordig te koop is, kenmerkt AUTOCAD zich door drie punten:

- AUTOCAD bevindt zich in het zgn. *low-end* van de CAD markt. Dat betekent dat AUTOCAD niet zo geavanceerd is als de vele uitgebreide systemen die zich in het *high-end* van deze markt bevinden. De prijs is echter ook bescheiden;
- AUTOCAD richt zich niet op één speciaal deel van de CAD markt. In feite is AUTOCAD een algemeen bruikbaar tekenpakket. AUTOCAD kent een open struktuur, waardoor het eenvoudig te *fine-tunen* is voor gebruik binnen een bepaald vakgebied (werktuigbouw, bouwkunde, etc.);
- AUTOCAD is (mede door zijn relatief lage prijs en zijn open struktuur) één van de meest verbreide CAD pakketten ter wereld. Hierdoor is AUTOCAD een interessant objekt geworden voor *third-party software developers*; bedrijfjes die speciale applikatie software schrijven voor AUTOCAD. Voor wie zelf geen toepassingsgerichte software wil of kan schrijven, is er dan ook een grote hoeveelheid applikatie software te koop.

Uiteraard gaan wij in deze AUTOLISP handleiding niet in op het gebruik van AUTOCAD zelf. De lezer wordt bekend verondersteld met tenminste de beginselen van het gebruik van AUTOCAD. Voor literatuur op het gebied van AUTOCAD verwijzen wij, behalve naar natuurlijk de *reference manual* [AC 88], naar o.a. [Gu 88, Oo 89, Ot 89, Sm 88, Th 88].

4 LISP en AutoLISP

4.1 LISP

LISP is een programmeertaal die reeds in de jaren 1956-62 werd ontwikkeld door de wiskundige John McCarthy tijdens zijn verblijf aan het Massachusetts Institute of Technology [Mc 62]. Samen met enkele van zijn studenten ontwikkelde McCarthy een nieuwe programmeertaal, die in eerste instantie bedoeld was voor symbolische data processing. Het voornaamste verschil tussen LISP en imperatieve programmeertalen als C, Pascal en BASIC (waarmee je wellicht bekend bent) is, dat LISP een funktionele programmeertaal is. Alles in LISP werkt met funkties en lijsten. Vandaar ook de naam; LISP staat voor LISt Processing.

LISP is met name een populaire programmeertaal in de wereld van expert systemen en kunstmatige intelligentie. In tegenstelling tot programmeertalen als C en Pascal werkt LISP meestal niet met een compiler, maar met een interpreter. Dat betekent dat tijdens exekutie een programma regel voor regel wordt vertaald in voor de komputer begrijpbare instrukties. Over het algemeen zijn programma's die met een interpreter werken langzamer dan gecompileerde programma's, omdat het vertalen tijdens de exekutie tijd kost. Om dit probleem te verhelpen zijn later ook compilers voor LISP programma's ontwikkeld.

In de loop der jaren zijn er heel veel verschillende versies van LISP uitgebracht. We spreken in dit verband wel van LISP dialekten. Enkele daarvan zijn: Common LISP, FranzLISP, InterLISP, MacLISP, UCI LISP en ZetaLISP. De eerst genoemde, Common LISP, is tegenwoordig de algemeen aanvaarde LISP standaard.

Voor meer informatie over de programmeertaal LISP en het gebruik daarvan verwijzen we graag naar de ruim voorradige literatuur op dat gebied [Mc 62, Ma 72, Wi 89, St 90]. Overigens, voor het lezen en begrijpen van deze AUTOLISP handleiding is voorkennis van funktioneel programmeren of LISP niet noodzakelijk.

4.2 LISP en AutoLISP

AUTOLISP is de programmeertaal waarin applikaties voor AUTOCAD geschreven kunnen worden. AUTOLISP is gebaseerd op Common LISP. De syntax van AUTOLISP komt sterk overeen met die van Common LISP. Echter: AUTOLISP kent slechts een deel van alle Common LISP konstrukties en bevat daarnaast een groot aantal standaard funkties die specifiek zijn voor gebruik binnen AUTOCAD. Vandaar ook dat AUTOLISP geen programmeertaal is zoals (Common) LISP, C, Pascal of BASIC; het is speciaal ontworpen voor gebruik binnen AUTOCAD.

AUTODESK, de maker van AUTOCAD, zegt bij de keuze van een applikatie programmeertaal voor LISP gekozen te hebben, omdat het één van de meest eenvoudig te leren programmeertalen zou zijn [AL 88]. Met name voor AUTOCAD-gebruikers die geen of nauwelijks programmeerervaring hebben, zou dit een voordeel zijn. Wij zetten vraagtekens bij dit argument. Funktioneel programmeren in een taal als AUTOLISP is fundamenteel anders dan het programmeren in een imperatieve taal. Veel gebruikers van AUTOCAD komen uit een technisch-wetenschappelijke omgeving. Enige kennis van programmeren in een imperatieve taal is veelal aanwezig. Om beter aan te sluiten bij deze voorkennis, had de programmeertaal van AUTOCAD wellicht beter op Pascal of BASIC gebaseerd kunnen zijn.

Andere argumenten van AUTODESK voor de keuze voor LISP zijn het feit dat een LISP interpreter eenvoudig en klein is (zeker voor systemen met beperkte geheugenkapaciteit een voordeel) en dat LISP uitermate geschikt is om te werken met verzamelingen heterogene objekten van verschillende omvang.

5 AutoLISP

In dit hoofdstuk worden de beginselen van het gebruik van AUTOLISP uitgelegd.

5.1 Eenvoudige rekenkundige funkties

AUTOLISP kommando's kunnen worden opgegeven na de *command prompt* van AUTOCAD. AUTOCAD herkent een kommando als een AUTOLISP kommando, als het omgeven is door haakjes. Een kommando zonder haakjes wordt verondersteld een kommando van AUTOCAD zelf te zijn.

Als eenvoudig voorbeeld laten we zien hoe je mathematische berekeningen kunt uitvoeren in AUTOLISP:

```
Command:  (+ 2.56 3.89)
6.45
```

Dit AUTOLISP kommando (het wordt tenslotte omgeven door haakjes en wordt daarom door AUTOCAD doorgegeven aan de AUTOLISP interpreter) vraagt om het resultaat van de optelling van de twee getallen 2.56 en 3.89. Het resultaat, 6.45, wordt door de komputer op het scherm afgedrukt.

Alle funkties in AUTOLISP worden gegeven in de zgn. *prefix notatie.* Dit betekent dat eerst de funktienaam wordt gegeven, gevolgd door de eventuele argumenten. In het voorbeeld hierboven is de funktie de optelling, de funktienaam is + en de argumenten zijn 2.56 en 3.89. Vanzelfsprekend is de optelling niet de enige funktie binnen AUTOLISP. Ook de andere standaard rekenkundige funkties (aftrekken, vermenigvuldigen en delen) zijn beschikbaar.

Funkties kunnen genest worden. Het resultaat van een funktieaanroep wordt dan weer gebruikt als argument van een andere funktie. Een voorbeeld hiervan is een aftrekking, waarvan het resultaat later in een vermenigvuldiging wordt gebruikt:

```
Command:  (* 3 (- 4.32 1.5))
8.46
```

AUTOLISP geeft altijd alleen het resultaat van de laatste funktieaanroep op het scherm weer. Vandaar dit in het bovenstaande voorbeeld alleen het eindresultaat 8.46 op het scherm verschijnt, en niet het tussenresultaat 2.82.

5.2 Geheugenvariabelen

Het is niet altijd mogelijk met alleen het nesten van funkties het gewenste resultaat te bereiken. Soms wil je het resultaat van een berekening opslaan, om het daarna meerdere keren te kunnen gebruiken. Zoals alle programmeertalen maakt AUTOLISP hiervoor gebruik van geheugen variabelen.

Geheugenvariabelen hebben namen die door de gebruiker zelf gekozen zijn. Deze namen bestaan uit een aaneengesloten reeks karakters (letters en cijfers, geen spaties). Een variabelenaam moet altijd beginnen met een letter. Kies ter wille van het geheugengebruik van de komputer niet al te lange namen. Omdat je als gebruiker zelf de namen van je variabelen kunt bepalen, kun je ze zó kiezen, dat de naam informatie bevat over de waarde van de variabele. Een hoek kan bijvoorbeeld `angle1` heten. Variabelen hoeven niet (zoals in sommige andere programmeertalen) gedeklareerd te worden; op het moment dat je een variabele voor de eerste keer gebruikt, wordt er door het systeem automatisch geheugenruimte voor gereserveerd.

Je kan een waarde aan een variabele toekennen met behulp van de standaard AUTOLISP funktie `setq`. Voorbeeld:

```
Command:  (setq x 2)
2
```

In bovenstaand voorbeeld wordt, indien de geheugenvariabele `x` nog niet eerder gebruikt was, een variabele `x` gekreëerd. Vervolgens wordt aan `x` de waarde 2 toegekend. Later kan deze variabele worden gebruikt in een andere funktieaanroep, bijvoorbeeld een deling:

```
Command:  (/ x 2)
1
```

Hier wordt de waarde van de variabele `x` door 2 gedeeld.
Een eenmaal gekreëerde variabele kan van waarde veranderen door er opnieuw een `setq` funktie op uit te oefenen:

```
Command:  (setq x 3)
3
```

Variabele `x` heeft nu de waarde 3 gekregen.
Je kan snel de waarde van een variabele inspekteren, door als kommando een uitroepteken, gevolgd door de variabele naam op te geven. Voorbeeld:

```
Command:  !x
3
```

Het uitroepteken is geen AUTOLISP funktie, maar een AUTOCAD kommando. Het uitroepteken kan ook worden gebruikt om waarden van systeemvariabelen van AUTOCAD te inspekteren. Vandaar dat in dit voorbeeld ook geen haakjes worden gebruikt.

5.3 Systeemvariabelen

AUTOCAD kent vele systeemvariabelen die informatie bevatten over de huidige tekening en zijn omgeving. Een voorbeeld van een systeemvariabele is **ORTHOMODE**, een systeemvariabele die aangeeft of AUTOCAD al dan niet in orthogonale mode werkt. Als **ORTHOMODE** de waarde 0 heeft, is de orthogonale mode uit, de waarde 1 geeft een ingeschakelde orthogonale mode aan. Een kompleet overzicht van deze systeemvariabelen kan worden gevonden in [AC 88].

In AUTOLISP is het mogelijk deze systeemvariabelen te inspekteren, dan wel te veranderen. Hiervoor maak je gebruik van de standaard AUTOLISP funkties **getvar** en **setvar**. Voorbeeld:

```
Command:  (getvar "ORTHOMODE")
0
Command:  (setvar "ORTHOMODE" 1)
1
Command:  (getvar "ORTHOMODE")
1
```

In het bovenstaande voorbeeld is achtereenvolgens de waarde van **ORTHOMODE** geïnspekteerd, 0 bevonden, vervolgens veranderd in de waarde 1, en tenslotte weer geïnspekteerd.

5.4 AutoLISP programma's schrijven en inladen

Een AUTOLISP programma wordt gevormd door de aanroep van één of meerdere funkties. Zo'n funktieaanroep kan (genest) ook weer aanroepen van andere funkties bevatten. In de loop van de uitvoering van het programma kunnen geheugen- en systeemvariabelen worden veranderd van waarde. Het resultaat van het programma is het resultaat van de laatste funktie aanroep.

Als je AUTOLISP programma's enige omvang krijgen, zul je ze niet meer op de

kommando regel van AUTOCAD in willen voeren. Ten eerste omdat dit onhandig werkt, maar vooral omdat je bij herhaalde exekutie van je programma telkens opnieuw dat programma moet intikken. Er bestaat dan ook een mogelijkheid om AUTOLISP programma's te schrijven en die daarna (zovaak als je dat wilt) te laden in AUTOCAD. De procedure hiervoor is als volgt:

1. start (buiten AUTOCAD) de tekst editor van je komputer op. Maak hiermee een file die de tekst van je AUTOLISP programma bevat;
2. schrijf deze file weg in de AUTOCAD-directory. De filenaam kun je vrij kiezen. De extensie van de filenaam moet echter `.lsp` zijn. Een geldige filenaam zou dus bijvoorbeeld `example.lsp` kunnen zijn;
3. start nu AUTOCAD op. Laad het AUTOLISP programma nu in AUTOCAD. Voorbeeld:

   ```
   Command: (load "example")
   ```

   AUTOCAD verondersteld nu dat de file `example.lsp` zich bevindt in de AUTOCAD-directory. Deze file wordt ingeladen en alle expressies in deze file worden geëvalueerd. Als de gezochte file zich niet in de AUTOCAD-directory bevindt, is het ook mogelijk het pad daarbij op te geven. Als de file `example.lsp` zich bevindt in de directory `C:\ACAD\LISP`, wordt het bijbehorende kommando:

   ```
   Command: (load "c:/acad/lisp/example")
   ```

   Merk op dat de *backslashes* uit MS-DOS zijn veranderd in *slashes* in het AUTOLISP kommando. De *backslash* heeft namelijk een speciale betekenis binnen AUTOLISP.

Programma's die op deze manier in AUTOCAD worden geladen, blijven in het geheugen van de komputer staan, totdat AUTOCAD beëindigd wordt. Als veel of grote programma's achter elkaar ingeladen worden, kan dit leiden tot een geheugentekort (zie ook Hoofdstuk 8).

5.5 Data types

AUTOLISP kent verschillende types om gegevens in op te slaan. Hieronder volgt een korte beschrijving.

5.5.1 Integers en reals

AUTOLISPkent twee data typen voor numerieke data. *Integers* zijn de gehele getallen tussen -32768 en 32767. Het gebruik van een integer buiten dit bereik resulteert in een foutmelding. Door hun beperkte bereik, gaat het rekenen met integers heel

snel. Een rekenkundige operatie (zoals optellen) met twee integers levert ook weer een integer op.

```
Command: (/ 7 2)
3
```

Het resultaat van een deling van de integer 7 door de integer 2 is dan ook 3 en niet 3.5.

Reals representeren ook waarden achter de decimale punt. Ze kunnen dus meer precisie geven geven tijdens berekeningen, maar operaties op reals zijn dientengevolge ook langzamer. Als we bovenstaand voorbeeld nog een keer willen uitrekenen, maar nu met reals, wordt dit

```
Command: (/ 7.0 2.0)
3.5
```

Het is eenvoudig om integers van reals te onderscheiden; integers bevatten nooit een decimale punt, reals altijd (zelfs als dat voor de waarde niet nodig zou zijn, zoals in het bovenstaande voorbeeld). Reals met een waarde kleiner dan 1 krijgen altijd een *leading zero*: .5 is geen geldige AUTOLISP waarde, 0.5 wel. Integers en reals kunnen ook gemengd worden tijdens een funktieaanroep. Om nogmaals hetzelfde voorbeeld te gebruiken:

```
Command: (/ 7.0 2)
3.5
```

De real 7.0 wordt nu gedeeld door de integer 2. Het resultaat is (zoals altijd wanneer reals en integers gekombineerd worden) een real.

5.5.2 Strings

Een *string* is een rijtje karakters (letters, cijfers, leestekens) die niet wiskundig bewerkt moeten worden. Een string wordt altijd omsloten door een aanhalingstekens. Strings worden meestal gebruikt om teksten in op te slaan die later voor de gebruiker zichtbaar moeten worden gemaakt. Strings kunnen elke willekeurige lengte hebben; de komputer reserveert zoveel geheugenruimte als nodig om ze op te slaan. Veel en lange strings kosten echter veel komputer geheugen en vertragen de uitvoering van het programma.

5.5.3 Lijsten

Een *lijst* in AUTOLISP is een rijtje van nul of meer lijstelementen. Een lijstelement kan alles zijn: een integer, een real, een string, een andere lijst, etc. Een lijst wordt altijd omsloten door een haakjespaar, terwijl de lijstelementen gescheiden worden door tenminste één spatie. Lijsten kunnen elke lengte hebben. Een lijst ter lengte nul is de lege lijst. Lijsten en funktioneel programmeren zijn onlosmakelijk met elkaar verbonden. In elke funktionele taal (zoals LISP en dus ook AUTOLISP) is de lijst een zeer belangrijk datatype.

AUTOLISP ziet x,y- dan wel x,y,z-koördinaten als lijsten met twee, respektievelijk drie elementen. De waarden van de koördinaten worden dus gescheiden door een spatie, zoals in onderstaand voorbeeld:

```
(4.36 2.67)
```

5.5.4 Entity namen

Elke tekening in AUTOCAD is opgebouwd uit één of meerdere *entities*. Een entity wordt beschreven door zijn *attributen*. Een rechte lijn is bijvoorbeeld een entity, die als attributen heeft zijn beginpunt, eindpunt en tekenlaag lokatie. Ingewikkelder entities hebben meer attributen.
AUTOLISP heeft funkties om een entity te selekteren en te gebruiken in een programma. Alle attributen worden daarbij dan als een lijst uit de teken database doorgegeven aan AUTOLISP. Het eerste element van de lijst is de naam van de entity. Hiervoor is een speciaal datatype gereserveerd.

5.5.5 File descriptors

AUTOLISP heeft funkties om files op disk te openen en er informatie uit te lezen. Eén van die funkties zagen we reeds in Paragraaf 5.4. Als een file wordt geopend, wordt hij toegekend aan een speciaal datatype, de zgn. *file descriptor*, die een pointer bevat naar het fysieke adres van de file op de disk.

5.6 Zelf funkties definiëren

Tot nu toe hebben we gesproken over *standaard* AUTOLISP funkties. Hiermee werden bedoeld funkties die altijd in AUTOLISP aanwezig zijn. Een beperkt overzicht

van deze funkties is te vinden in Appendix A. Een volledig overzicht van alle standaard funkties is te vinden in [AL 88].

Voor een gebruiker is het echter ook mogelijk zelf nieuwe funkties te definiëren. Deze funkties kunnen dan opgebouwd zijn uit bestaande standaard funkties of uit andere zelfgedefinieerde funkties. Alles binnen de funktionele taal AUTOLISP gaat met funkties; zo ook het definiëren van nieuwe funkties. Hiervoor maken we gebruik van de standaard funktie **defun**. **defun** heeft twee argumenten: de naam van de nieuw te definiëren funktie (kompleet met zijn argumenten) en een beschrijving van de nieuwe funktie, uitgedrukt in één of meerdere andere funkties (die zowel standard AUTOLISP funkties kunnen zijn, als eerder zelf gedefinieerde funkties).

Als voorbeeld definiëren we een funktie die bij een door de gebruiker gegeven straal (d.m.v. twee punten) de omtrek van de bijbehorende cirkel berekend.

```
(defun omtrek ()
  ; bepaal eerst het middelpunt van de cirkel
  (setq pt1
        (getpoint "\nGeef het middelpunt van de cirkel: ")
  )
  ; geef een punt op de cirkel om later de straal
  ; te kunnen bepalen
  (setq pt2
        (getpoint "\nGeef een punt op de cirkel: ")
  )
  ; bepaal de straal (afstand tussen twee punten)
  ; en vervolgens de omtrek
  (* 2
     (* pi (distance pt1 pt2))
  )
)
```

Het bovenstaande is in feite een klein AUTOLISP programma, waarin met behulp van **defun** de funktie **omtrek** wordt gedefinieerd. Het programma is voorzien van *kommentaar*; regels die niet voor de komputer, maar slechts voor een menselijke lezer van het programma bestemd zijn. AUTOLISP beschouwt alles wat na een punt-komma komt tot het einde van die regel als kommentaar. Verder zie je dat het programma is opgeschreven in een bepaalde layout. Dit is ook gedaan om het programma beter leesbaar te maken voor mensen. Door bij elkaar horende haakjes onder elkaar te plaatsen, zie je beter of je geen haakjes vergeten bent. Bij eenvoudige funkties is het aantal haakjes nog wel te overzien, maar je zult zien dat dit bij ingewikkelder programma's snel oploopt (dit is overigens een typisch kenmerk van alle LISP-achtige talen).
De naam van de funktie die we hier definiëren is **omtrek**. Deze funktie heeft geen argumenten, vandaar dat we een lege argumentlijst meegeven (()). De funktie

getpoint is een standaard AUTOLISP funktie, die de meegegeven string afdrukt op het scherm en vervolgens pauzeert totdat de gebruiker een punt heeft opgegeven. De gebruiker kan dit doen door x,y-koördinaten op te geven via het toetsenbord, maar ook het aangeven van een punt d.m.v. een muis of *digitizer* wordt als invoer geaksepteerd. Het resultaat van deze funktie is het opgegeven punt. Deze informatie wordt met de **setq** funktie van AUTOLISP toegekend aan variabele **pt1**. Evenzo wordt een tweede punt aan de gebruiker gevraagd en dit punt wordt toegekend aan de variabele **pt2**. **distance** is een standaard AUTOLISP funktie die de afstand tussen twee punten (in dit geval **pt1** en **pt2**) berekent. Deze afstand wordt beschouwd als de straal van de cirkel. Vervolgens wordt de omtrek van de cirkel berekend volgens de bekende formule

$$\text{omtrek} = 2 \times \pi \times \text{straal} .$$

Zoals je ziet worden er in dit voorbeeld diverse keren funkties genest. De niet geneste funkties worden achter elkaar uitgevoerd. Het uiteindelijke resultaat is het resultaat van de laatste funktie aanroep, in dit geval de buitenste vermenigvuldiging. Dit wordt nu het resultaat van de funktie **omtrek**. Overigens, π is een konstante van AUTOLISP. Als je met π wil rekenen, is het beter deze konstante te gebruiken in plaats van zelf een real met die waarde te definiëren, omdat de konstante π altijd nauwkeuriger is. AUTOLISP rekent, als het om hoeken gaat, altijd met radialen. Graden moeten dus altijd eerst worden omgezet in radialen (vermenigvuldigen met een faktor $\pi/180$ - zie ook het voorbeeld in 6).

5.7 Koppeling naar AutoCAD

5.7.1 Definiëren van nieuwe AutoCAD kommando's

De funktie **defun** kan worden gebruikt om nieuwe AUTOLISP funkties te definiëren. Op dezelfde manier kunnen echter ook nieuwe AUTOCAD kommando's worden gedefinieerd. De naam van de funktie moet dan worden voorafgegaan door **C:**. Hierna kan de nieuwe funktie als een AUTOCAD kommando worden gebruikt. Dit betekent o.a. dat een aanroep ervan niet meer hoeft te zijn omsloten door een haakjespaar.

5.7.2 Aanroepen van AutoCAD kommando's in AutoLISP

Om kommando's van AUTOCAD te kunnen laten uitvoeren door een AUTOLISP programma, bestaat de standaard AUTOLISP funktie **command**. Deze funktie verwacht als parameters een AUTOCAD kommando en (eventueel) de bijbehorende parameters. Zo kent AUTOCAD het kommando **LINE** dat een lijnstuk tekent van een door de gebruiker op te geven beginpunt tot een, eveneens door de gebruiker

op te geven, eindpunt. Als we dit kommando in een AUTOLISP programma willen laten uitvoeren, dan gaat dat als volgt:

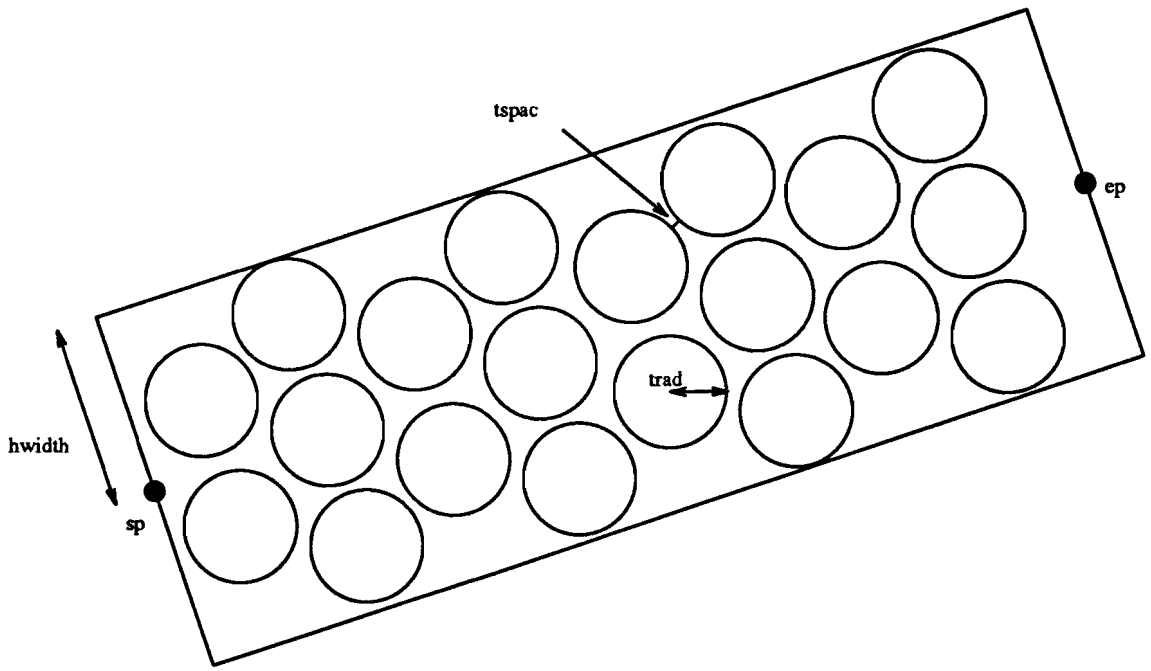
```
(command "LINE" "0,0" "5,5" "")
```

Alle parameters van de funktie **command** moeten als strings (dus tussen aanhalingstekens) worden opgegeven. Dit is noodzakelijk, omdat de funktie **command** niets anders doet dan zijn parameters letterlijk doorgeven aan AUTOCAD. Het laatste paar aanhalingstekens, die niets omsluiten, zijn equivalent met het indrukken van de **return**-toets op het toetsenbord.

6 AutoLISP voorbeeld

Om de mogelijkheden die AUTOLISP biedt te verduidelijken en de werkwijze met AUTOLISP te tonen, behandelen we in dit hoofdstuk een voorbeeld van een AUTOLISP programma. Dit voorbeeld is ontleend aan [AL 88].

Stel men wil een rechthoekig tuinpad kunnen kreëren van ronde tegels. De struktuur van het pad is weergegeven in Figuur 6. Hierbij moeten de grootte en de plaats van het pad en de straal van de tegels en de afstand tussen de tegels variabel zijn.



Figuur 1: Het tuinpad met parameters

De gebruiker moet de volgende parameters invoeren: het beginpunt van het pad (`sp`), het eindpunt van het pad (`ep`), de halve breedte van het pad (`hwidth`), de straal van de tegels (`trad`) en de afstand tussen de tegels (`tspac`).

Start de tekst editor van de komputer op en typ daar onderstaand programma in zoals aangegeven in 5.4. Voor de komputer (of, om precies te zijn de AUTOLISP interpreter) hoef je daarbij niet precies alle regels en karakters te plaatsen zoals dat hier gebeurd is. Echter, inspringen en af en toe regels openlaten is wel wenselijk,

omdat dit de leesbaarheid van het programma vergroot. Op verschillende plaatsen is kommentaar tussengevoegd. Na elke puntkomma is de rest van de regel kommentaar (d.w.z. behoort niet tot de programma tekst en wordt door de interpreter niet gelezen, doch is slechts bedoeld voor de man of vrouw die de programmatekst bekijkt). Kommentaar is bedoeld om de leesbaarheid van het programma vergroten, zodat bijvoorbeeld anderen kunnen zien wat jouw programma doet.

```
;    Convert angle in degrees to radians

(defun dtr (a)
     (* pi (/ a 180.0))
)

;       Acquire information for garden path

(defun gpuser ()
        (setq sp (getpoint "\nStart point of path: "))
        (setq ep (getpoint "\nEnd point of path: "))
        (setq hwidth (getdist "\nHalf width of path: " sp ))
        (setq trad (getdist "\nRadius of tiles: " sp))
        (setq tspac (getdist "\nSpacing between tiles: " sp))

        (setq pangle (angle sp ep))
        (setq plength (distance sp ep))
        (setq width (* 2 hwidth))
        (setq angp90 (+ pangle (dtr 90)))  ; Path angle + 90 deg
        (setq angm90 (- pangle (dtr 90)))  ; Path angle - 90 deg
)

;       Draw outline of path

(defun drawout ()
        (command "pline"
             (setq p (polar sp angm90 hwidth))
             (setq p (polar p pangle plength))
             (setq p (polar p angp90 width))
             (polar p (+ pangle (dtr 180)) plength)
             "close"
        )
)

;       Place one row of tiles given distance along path
;       and possibly offset it

(defun drow (pd offset)
        (setq pfirst (polar sp pangle pd))
        (setq pctile (polar pfirst angp90 offset))
        (setq p1tile pctile)
        (while (< (distance pfirst p1tile) (- hwidth trad))
```

```
            (command "circle" p1tile trad)
            (setq p1tile (polar p1tile angp90 (+ tspac trad trad)))
        )
        (setq p1tile (polar pctile angm90 (+ tspac trad trad)))
        (while (< (distance pfirst p1tile) (- hwidth trad))
            (command "circle" p1tile trad)
            (setq p1tile (polar p1tile angm90 (+ tspac trad trad)))
        )
)

;       Draw the rows of tiles

(defun drawtiles ()
        (setq pdist (+ trad tspac))
        (setq off 0.0)
        (while (<= pdist (- plength trad))
            (drow pdist off)
            (setq pdist (+ pdist (* (+ tspac trad trad) (sin (dtr 60)))))
            (if (= off 0.0)
                (setq off (* (+ tspac trad trad) (cos (dtr 60))))
                (setq off 0.0)
            )
        )
)

;       Execute command, calling constituent functions

(defun C:PATH ()
        (gpuser)
        (drawout)
        (drawtiles)
)
```

Schrijf het programma nu naar disk, door het te *saven* als een file met de naam `gp.lsp`. Denk er hierbij aan, dat het programma weggeschreven moet worden als een ASCII file. Kopiëer de file (indien nodig) naar de directory van waaruit men AUTOCAD opstart. Start dan AUTOCAD op en typ het volgende op de kommando regel:

```
Command:  (load "gp")
```

In het programma komt zes keer de funktie `defun` voor. Alle funkties die door middel van deze funktie worden gedefiniëerd kunnen binnen AUTOCAD gebruikt worden. In de volgende sekties wordt elk van deze funkties kort behandeld. De funkties gebruiken zelf weer andere (standaard) AUTOLISP funkties. Deze worden hier niet in detail behandeld. Hiervoor verwijzen we naar [AL 88].

6.1 Funktie dtr

De funktie **dtr** rekent graden om naar radialen. AUTOLISP werkt intern niet met graden, maar met radialen. Als we de gebruiker toch waarden willen laten kunnen opgeven in graden, hebben we deze konversie funktie nodig.

```
;    Convert angle in degrees to radians

(defun dtr (a)
     (* pi (/ a 180.0))
)
```

De funktie **dtr** heeft een argument (a, dat staat voor *angle*), voorstellende de hoek waarvoor de omrekening moet plaats vinden. We kunnen **dtr** testen door bijvoorbeeld in te typen:

```
Command:  (dtr 180)
3.14159
```

Een hoek van 180 graden komt inderdaad overeen met een hoek van π radialen.

6.2 Funktie gpuser

De funktie **gpuser** vraagt de gebruiker de gewenste variabelen een waarde te geven en berekent vervolgens nog enkele andere variabelen.

```
;       Acquire information for garden path

(defun gpuser ()
        (setq sp (getpoint "\nStart point of path: "))
        (setq ep (getpoint "\nEnd point of path: "))
        (setq hwidth (getdist "\nHalf width of path: " sp ))
        (setq trad (getdist "\nRadius of tiles: " sp))
        (setq tspac (getdist "\nSpacing between tiles: " sp))

        (setq pangle (angle sp ep))
        (setq plength (distance sp ep))
        (setq width (* 2 hwidth))
        (setq angp90 (+ pangle (dtr 90)))  ; Path angle + 90 deg
        (setq angm90 (- pangle (dtr 90)))  ; Path angle - 90 deg
)
```

We kunnen de funktie **gpuser** uitproberen door in te typen:

```
Command: (gpuser)
```

AUTOCAD kijkt nu of de funktie **gpuser** bekend is en voert hem dan uit. Dit betekent dus dat er om invoer gevraagd wordt. Dit wordt gedaan met behulp van de (standaard) AUTOLISP funkties **getpoint** en **getdist**. De ingevoerde waarden worden door gebruik van de funktie **setq** aan de variabelen toegekend. Antwoord op de vragen bijvoorbeeld het volgende:

```
Start point of path: 20,20
End point of path: 90,80
Half width of path: 20
Radius of tiles: 2
Spacing between tiles: 1
```

Om punten in te geven, kan men, in plaats van via het toetsenbord de koördinaten op te geven, ook de muis of digitizer gebruiken om punten aan te klikken. **gpuser** geeft de variabelen die gebruikt worden hun passende waarde. Zo wordt bijvoorbeeld de lengte van het pad (**plength**) met **setq** gezet op de afstand (de funktie **distance**) tussen het beginpunt (variabele **sp**) en het eindpunt (variabele **sp**). Op het scherm verschijnt het resultaat van de laatste funktieaanroep (hier dus de waarde van angm90).

Om te kontroleren of alles goed gegaan is, kan men de waarde van een variabele opvragen. Voorbeeld:

```
Command: !hwidth
20
```

6.3 Funktie drawout

De funktie **drawout** tekent de kontouren van het pad op het scherm. Hiervoor wordt de funktie **command** (zie Paragraaf 5.7.2) gebruikt. Met de **command** funktie worden AUTOCAD tekenfunkties aangeroepen. Hier wordt "**pline**" gebruikt. "**pline**" tekent de kontouren van het pad als een polylijn. Hiervoor wordt de funktie **polar** (een standaard AUTOLISP funktie) gebruikt, die als argumenten het startpunt, de hoek en de lengte van de lijn heeft. "**close**" zorgt er voor dat begin- en eindpunt van de polylijn met elkaar verbonden worden.

```
;       Draw outline of path

(defun drawout ()
       (command "pline"
```

```
            (setq p (polar sp angm90 hwidth))
            (setq p (polar p pangle plength))
            (setq p (polar p angp90 width))
            (polar p (+ pangle (dtr 180)) plength)
            "close"
        )
)
```

Om het resultaat te kunnen zien moet men simpelweg het volgende intypen:

```
Command: (drawout)
```

Alvorens **drawout** aan te roepen, moet natuurlijk eerst wel al een keer **gpuser** aangeroepen zijn geweest, omdat de variabelen die in **drawout** gebruikt worden anders nog geen waarde hebben

6.4 Funktie drow

De funktie **drow** tekent een rij tegels en heeft twee argumenten. Het eerste argument is de afstand van de rij tot het beginpunt sp. Het tweede argument geeft aan of de rij in het midden een tegel heeft of niet. Dit wordt gedaan door de afstand van de eerste te tekenen tegel vanaf de middellijn mee te geven.

```
;       Place one row of tiles given distance along path
;       and possibly offset it

(defun drow (pd offset)
        (setq pfirst (polar sp pangle pd))
        (setq pctile (polar pfirst angp90 offset))
        (setq p1tile pctile)
        (while (< (distance pfirst p1tile) (- hwidth trad))
             (command "circle" p1tile trad)
             (setq p1tile (polar p1tile angp90 (+ tspac trad trad)))
        )
        (setq p1tile (polar pctile angm90 (+ tspac trad trad)))
        (while (< (distance pfirst p1tile) (- hwidth trad))
             (command "circle" p1tile trad)
             (setq p1tile (polar p1tile angm90 (+ tspac trad trad)))
        )
)
```

De variabele **p1tile** is altijd het middelpunt van de volgende te tekenen tegel. In de eerste **while** lus worden de tegels aan de ene kant van de middellijn getekend en in de tweede lus die aan de andere kant.

6.5 Funktie drawtiles

De functie **drawtiles** zorgt ervoor dat de goede rijen tegels getekend worden. In de **while**-lus wordt elke keer de positie langs de middellijn van de volgende te tekenen rij bepaald en wordt de afstand van de eerste tegel in die rij tot de middellijn berekend. Deze waarde komt in de variabele **off** te staan.

```
;       Draw the rows of tiles

(defun drawtiles ()
        (setq pdist (+ trad tspac))
        (setq off 0.0)
        (while (<= pdist (- plength trad))
             (drow pdist off)
             (setq pdist (+ pdist (* (+ tspac trad trad) (sin (dtr 60)))))
             (if (= off 0.0)
                   (setq off (* (+ tspac trad trad) (cos (dtr 60))))
                   (setq off 0.0)
             )
        )
)
```

Typ het volgende:

```
Command:  (drawtiles)
```

en zie daar: de tegels worden netjes voor je gelegd (natuurlijk weer onder de voorwaarde dat je eerst **gpuser** en **drawout** hebt aangeroepen).

6.6 Kommando PATH

```
;       Execute command, calling constituent functions

(defun C:PATH ()
        (gpuser)
        (drawout)
        (drawtiles)
)
```

Het einde van het programma in de file **gp.lsp** wordt gevormd door de definitie van **C:PATH**. Hierdoor wordt er aan de reeks van AUTOCAD kommando's een nieuw kommando **PATH** toegevoegd. Het snel en gemakkelijk invoegen van tuinpaden in je ontwerp in AUTOCAD komt nu onder handbereik. Voorwaarde is dat de gekozen

naam (in dit geval PATH) nog geen AUTOCAD kommando is. Het nieuwe AUTOCAD kommando kan (net als andere kommando's) worden aangeroepen door zijn naam in te geven. PATH is een AUTOCAD kommando en géén AUTOLISP funktie en daarom hoeven er bij aanroep geen haakjes omheen geplaatst te worden.

```
Command: PATH
Start point of path: 20,20
End point of path: 90,80
Half width of path: 20
Radius of tiles: 2
Spacing between tiles: 1
```

Automatisch verschijnt nu een tuinpad met de goede parameters op het scherm van de komputer.

7 Bijgeleverde programma's

AUTOCAD wordt geleverd met een aantal AUTOLISP programma's waarmee men verscheidene figuren kan kreëren. Deze programma's staan op de zgn. *"Support"* disks en (indien aanwezig) op de *"Bonus"* disk. Men kan deze programma's inladen op de manier beschreven in Paragraaf 5.4. Om meer inzicht in het programmeren met AUTOLISP te krijgen, raden wij je aan om eens naar de programmateksten te kijken. Wij geven hier een paar voorbeelden.

In de file `3d.lsp` staan een aantal AUTOLISP funkties om drie-dimensionale objekten te kreëren. Deze file kan men binnenhalen door "3D Construction" te kiezen van het "DRAW" pull-down menu, "3D objects" te kiezen van het "3D" submenu of door te typen:

```
Command: (load "3d")
```

Het kommando "3D" is dan beschikbaar en heeft het volgende resultaat:

```
Command: 3D
Box/Cone/DIsh/DOme/Mesh/Pyramid/Sphere/Torus/Wedge:
```

`3d.lsp` definiëert elk van deze objekten ook als een kommando. We zullen deze hier niet één voor één gaan behandelen, maar een voorbeeld nemen. Stel dat we een kubus op het scherm willen dan gaat dit als volgt:

```
Command: BOX
Corner of box: geef een punt
Length: geef een afstand
Cube/<Width>: afstand of "C"
Rotation angle about Z axis: geef een hoek
```

Als men bij de voorlaatste regel "C" typt dan wordt er een kubus gemaakt. Als men een getal typt dan wordt dit als de breedte geïnterptreteerd en wordt er vervolgens nog om de hoogte gevraagd. In allebei de gevallen wordt er nog om de rotatie met de z-as gevraagd.
U hebt nu slechts een klein voorbeeld gezien van een standaard AUTOLISP programma. Een volledigere beschrijving van alle 3D-objekten vind men in [AL 88].

Er staan nog vele andere programma's op de "Support" disk. Andere leuke voorbeelden staan op de "Bonus" disk. Deze programma's zijn meer ter demonstratie bedoeld en niet zo volledig uitgewerkt. Een mooi voorbeeld is de file `fplot.lsp` waarin het `FPLOT` komando wordt gedefiniëerd. Hiermee kan men heel eenvoudig funktie van twee variabelen op het scherm uitplotten. Het werkt als volgt:

```
Command: (fplot <funktie> <xrange> <yrange> <resolutie>)
```

Een voorbeeld is:

```
(fplot  '(lambda (x y) (cos (sqrt (+ (* x x 2) (* y y)))))
'(-2 2)
'(-2 2)
20
)
```

Voor als men zelf geen funktie van twee variabelen kan verzinnen staat er in `fplot.lsp` ook een standaard voorbeeld:

```
Command: DEMO
```

Hiermee krijg je een prachtige demonstratie van de mogelijkheden die AUTOLISP en AUTOCAD op dit gebied te bieden hebben

8 Problemen

8.1 Ongebalanceerde haakjes

Het gebruik van een LISP-achtige programmeertaal gaat gepaard met het gebruik van grote aantallen haakjesparen. Hoe ingewikkelder de programma's en hoe *dieper* de nesting van funkties, hoe groter het aantal haakjes dat op het scherm verschijnt. AUTOLISP is daarop geen uitzondering.

Haakjesexpressies dienen *gebalanceerd* te zijn. Dat wil zeggen dat bij elk haakje dat geopend wordt, ook weer een haakje gesloten dient te worden. Het verschil tussen het aantal openingshaakjes en het aantal sluithaakjes dient tijdens een expressie steeds groter of gelijk aan nul te zijn, en na evaluatie van de expressie moet het weer nul zijn.

Het gebruik van haakjes is nogal foutgevoelig; het is gemakkelijk om een haakje te vergeten. De AUTOLISP-interpreter loopt daarop stuk en geeft een foutmelding van de vorm:

```
n>
```

Hierbij is n het aantal sluithaakjes dat mist om de geëvalueerde expressie gebalanceerd te laten zijn. De enige manier om van deze foutmelding af te komen, is middels het toetsenbord het gewenste aantal missende haakjes in te toetsen. Als na het ingeven van het gevraagde aantal sluithaakjes de prompt nog niet verdwijnt, is de fout vermoedelijk een missend aanhalingsteken. Hierdoor wordt alle tekst na het openen van de aanhaling als één grote string beschouwd. De oplossing is dan om eerst een aanhalingsteken in te tikken, gevolgd door het gevraagde aantal sluithaakjes.

Het is wenselijk om na het op deze manier provisorisch verhelpen van de fout, met behulp van de tekst editor eerst de kode van het AUTOLISP programma te verbeteren, omdat deze kennelijk een fout bevatte. De resultaten van het foute programma zijn dan ook onbetrouwbaar. Om programmeerfouten als deze te voorkomen, is het goed de aanwijzingen uit de hoofdstukken 5 en 6 met betrekking tot de layout van het programma op te volgen.

8.2 Stack- en heap size

AUTOLISP files worden volledig in het geheugen van de komputer geladen, voordat ze worden uitgevoerd. Om deze reden is het noodzakelijk dat er in het geheugen voldoende ruimte is om alle variabele namen, funkties, strings, enzovoorts op te

slaan. Als je verschillende AUTOLISP programma's achter elkaar in het geheugen laadt (of één heel groot programma), loop je de kans de volgende foutmelding te krijgen:

```
insufficient node space
```

Om meer ruimte te kreëeren voor de AUTOLISP funkties, moet je de volgende twee DOS kommando's opnemen in de file `AUTOEXEC.BAT`, die te vinden is in de *root directory*:

```
SET LISPSTACK = 10000
SET LISPHEAP = 35000
```

Je kan de waarde van deze twee (DOS) systeemvariabelen laten variëren; hun som mag echter nooit meer dan 45000 bedragen. `LISPHEAP` bepaalt hoeveel geheugenruimte er gereserveerd wordt voor funkties, geheugenvariabelen, e.d. `LISPSTACK` bepaalt hoe diep je funkties kunt nesten. Meestal wordt `LISPHEAP` als de grootste van de twee gedefinieerd, met name als je veel verschillende funkties en geheugenvariabelen gebruikt. Problemen met nesting kunnen echter worden opgelost door `LISPSTACK` te vergroten; mogelijk ten koste van `LISPHEAP`. Overigens, het zetten van deze systeemvariabelen heeft geen invloed op andere programma's dan AUTOCAD.

8.3 Geheugentekort

AUTOCAD en AUTOLISP zijn in feite aparte programma's, die tegelijkertijd in het werkgeheugen van de komputer zitten. AUTOCAD heeft een oplossing voor een tekort aan geheugenruimte. Grote tekeningen die normaal niet in het werkgeheugen zouden passen, worden automatisch gedeeltelijk *uitgeswapt* naar de disk. We spreken wel van *virtual memory*.

AUTOLISP kent deze voorziening niet. Daarom is AUTOLISP beperkt tot de 64K limiet van PC's, waarvan slechts 45K nuttig gebruikt kan worden voor *data*. Echter, onder data wordt hier verstaan alle programma's, hun variabelen en hun data. Een tekort aan geheugenruimte door in het geheugen akkumulerende programma's treedt dan ook al snel op. Door middel van de AUTOLISP funktie `vmon` (*Virtual Memory ON*) kan de virtueel geheugen faciliteit worden uitgebreid van alleen maar AUTOCAD naar AUTOCAD én AUTOLISP. Weinig gebruikte funktiedefinities worden nu tijdelijk uitgeswapt. Variabelen, strings en data worden echter nooit uitgeswapt. Derhalve blijft een zuinig omspringen hiermee vereist.

9 Konklusie

AUTOCAD is een veelverkocht en -gebruikt CAD-pakket. Het is een algemeen bruikbaar, niet specifiek op één toepassingsgebied gericht CAD-pakket. Het kent echter een open struktuur, waardoor het mogelijk is voor de gebruiker het pakket te *fine-tunen* naar zijn specifieke wensen en behoeften. Dit kan onder andere door het schrijven van zgn. *makro's*. AUTOLISP, de op Common LISP gebaseerde programmeertaal van AUTOCAD, is echter het ultieme middel om speciale applikatie software te maken. Door middel van AUTOLISP heeft de gebruiker de mogelijkheid nieuwe kommando's te definiëren en bestaande kommando's te herdefiniëren. Achter deze kommando's kunnen (al dan niet gekompliceerde) algoritmen en numerieke berekeningen schuil gaan.

Wij zetten vraagtekens bij de keuze van AUTODESK (de makers van AUTOCAD en AUTOLISP) voor LISP als de basis voor hun programmeertaal. Wellicht was een imperatieve gestruktureerde taal als Pascal een keuze geweest die beter had aangesloten bij de voorkennis van de gebruikers. Funktioneel programmeren is fundamenteel anders dan imperatief programmeren en maakt daardoor de drempel voor AUTOCAD gebruikers om zelf in AUTOLISP te gaan programmeren hoger. Desalniettemin menen wij dat deze handleiding een goed eerst begin is in de richting van het zelf ontwikkelen van AUTOLISP programma's en daarmee het *fine-tunen* van AUTOCAD. Daarnaast zijn er natuurlijk ook vele *third-party software developers*, die op kommerciële basis software in AUTOLISP ontwikkelen. Ook deze software kan een goede bron van aanvullingen zijn op het kale standaard CAD-pakket, dat AUTOCAD is.

De (toekomstige) programmeur van AUTOLISP programma's zij gewaarschuwd voor de te verwachten geheugen tekorten. De beperkte geheugenruimte die voor AUTOLISP routines en data ter beschikking staat, kan al snel tot problemen leiden, zodra de programma's enige omvang krijgen. Inventiviteit en grondiger systeemkennis zijn dan vereist om deze problemen op te lossen.

A Standaard funkties

Kategorie	Funktie	Syntax
Wiskunde	1+	(1+ number)
	1-	(1- number)
	abs	(abs number)
	cos	(cos angle)
	expt	(expt [base number][power])
	gcd	(gcd number number)
	sin	(sin angle)
	sqrt	(sqrt number)
User prompts	getangle	(getangle [ref point][prompt])
	getdist	(getdist [ref point][prompt])
	getint	(getint prompt)
	getpoint	(getpoint [refpoint][prompt])
	getreal	(getreal prompt)
	getstring	(getstring value prompt)
	prompt	(prompt ''message'')
Koördinaat	angle	(angle [first point][second point])
	car	(car list)
	cadr	(cadr list)
	caddr	(caddr list)
	distance	(distance [first point][second point])
	inters	(inters point point point point nil)
	list	(list argumentlist)
	mapcar	(mapcar 'function list list ...)
	osnap	(osnap [point][snap mode])
	polar	(polar [ref point][angle][distance])
Logisch	=	(= data data)
	/=	(/= data data)
	>	(> number number)
	<	(< number number)
	>=	(>= number number)
	<=	(<= number number)
	and	(and [logical function][logical function]...)
	equal	(equal data data)
	not	(not [logical function])
	or	(or [logical function][logical function])
Kontrole	if	(if [logical expression][function if true][function if false])
	while	(while [logical expression][function while true])
Strings	strcase	(strcase string upper/lower)
	strcat	(strcat string1 string2 ...)
	subst	(subst 'new 'old list)
	substr	(substr string start length)
Data type	atof	(atof string)
	atoi	(atoi number)
	fix	(fix real)
	float	(float integer)
	itoa	(itoa integer)
	rtos	(rtos string)
Speciaal	defun	(defun [name(arguments)][other functions])
	command	(command ["AutoCAD command"][command options and prompt responses])
	quote	(quote data)

Tabel 1: Enkele standaard funkties in AUTOLISP

Referenties

[AC 88] AUTODESK Ltd.,
"AutoCAD^R Release 10 - Reference Manual",
AUTODESK Ltd., London, GB, 1988

[AL 88] AUTODESK Ltd.,
"AutoLISP^R Release 10 - Programmer's Reference",
AUTODESK Ltd., London, GB, 1988

[Gu 88] Jeff Guenther, Ed Ocoboc, Anne Wayman,
"AutoCAD - Methods and Macros",
TAB Professional and Reference Books, Blue Ridge Summit, PA, 1988,
ISBN 0-8306-0189-9

[Ma 72] W.D. Maurer,
"The Programmer's Introduction to LISP",
Macdonald, London,
ISBN 0-444-19572-6

[Mc 62] John McCarthy e.a.,
"LISP 1.5 Programmer's Manual",
The M.I.T. Press, Cambridge, MA

[Oo 89] Walter van Oostveen, Milco Schmidt,
"Het AutoCAD boek",
Uitgeverij Sifra, Eeserveen, NL, 1989,
ISBN 90-6983-025-6

[Ot 89] Ad den Otter,
"Het AutoCAD Handboek - Release 10.0",
Addison-Wesley Publishing Company, Inc., Amsterdam, NL, 1989,
ISBN 90-6789-120-7

[Sm 88] Joseph Smith, Rusty Gesner,
"Customizing AutoCAD",
New Riders Publishing, Thousand Oaks, CA, 1988,
ISBN 0-934035-19-0

[St 90] W. Richard Stark,
"LISP, Lore, and Logic",
Springer-Verlag, New York, NY,
ISBN 0-387-97072-X

[Th 88] Robert M. Thomas,
"Advanced Techniques in AutoCAD - For Release 9",
Sybex, Alamede, CA, 1988,
ISBN 0-89588-437-2

[Wi 89] Patrick Henry Winston, Berthold Klaus Paul Horn,
"LISP - third edition",
Addison-Wesley Publishing Company, Reading, MA,
ISBN 0-201-08319-1

Index