AutoLISP-Perusteet

AutoLISP (myös Visual Lisp) on AutoDeskin kehittämä ohjelmointikieli AutoCADin toimintojen automatisoimiseen. Tämä tehostaa työtä poistamalla ihmiseltä aikaa vievää, tylsää ja toistuvaa työtä, jättäen aikaa tärkeämpään suunnitteluun.

AutoLISP on helposti lähestyttävä kieli, vaikka ulkonäkö on erikoinen: sulkuja, sulkuja ja sulkuja. Lisp on lyhenne sanoista List processing (listaprosessointi), mikä on kuvaavaa, sillä lispissä on vain yksi, monipuolinen tietorakenne: *lista*. Itse lisp-koodikin kirjoitetaan lispin ymmärtämässä listamuodossa.

Visual Lisp-kehitysympäristö VLIDE

Jotta pääsemme alkuun, on hyvä avata AutoCADiin Visual lisp-kehitysympäristö. Tämä onnistuu, kun menee "Manage"-välilehdellä osioon "Applications" ja klikkaa "Visual LISP-editor", tai kirjoittaa CADin komentoriville VLIDE. Tästä avautuu ikkuna, jonka sisällä on ikkuna Visual LISP Console. Tähän kirjoittamalla voi kokeilla komentojen toimintaa. Vasemmasta yläkulmasta saa avattua uuden tiedoston, johon voi alkaa kirjoittaa lisp-koodia*. Kun etsii apua komennon löytämiseen, F1 avaa AutoCADin help-ikkunan, josta voi seikkailla AutoLISP-reference -osioon, jossa kerrotaan kaikkien AutoLISPin komentojen toiminnasta. Täältä on myös pienen harjoittelun jälkeen helppo löytää sopiva komento ohjelmoinnin avuksi.

Tässä oppaassa käytetään esimerkkeinä Visual Lisp -konsoliin syötettyjä komentoja. Konsoli odottaa komentoa, kun sen rivin alussa on `_$`. Tämän perään voi kirjoittaa komennon, ja konsoli suorittaa komennon kun painetaan `enter`. Komennon palauttama arvo ilmestyy kirjoitetun rivin alle. Esimerkkinä syötetään komento `(+ 10 5)` konsoliin, jolloin seuraavalle riville ilmestyy palautettu arvo `15` ja konsoli ilmoittaa odottavansa seuraavaa komentoa uudella rivillä taas merkein `_$`:

```
_$ (+ 10 5)
15
_$
```

Konsoliin voi kirjoittaa useamman rivin ennen suoritusta, jos painaa `shift + enter` kun haluaa riviä. Esimerkeissä annan koodipätkiä, jotka voi myös itse kirjoittaa konsoliin. Nämä eivät välttämättä ole kaikkein käytännöllisimpiä esimerkkejä, mutta niistä näkee rivi riviltä, mitä lispissä tapahtuu.

Mitä tarkoittaa palauttaminen? Miten se toimii lispissä?

Palauttaminen tarkoittaa, että suoritettu komento/lause antaa kutsujalle jonkin arvon takaisin. Lispissä lause palauttaa aina sen viimeisimmän arvon. Tämän tarkoitus näkyy paremmin myöhemmissä esimerkeissä.

Lispin syntaksi eli kielioppi

AutoLISPin lauseet noudattavat yhtä kaavaa: `(komento parametri1 parametri2 ...)`. Sulkujen sisällä on ensin komento, ja sen jälkeen komennolle annetut parametrit, joilla komento suoritetaan. Esimerkkinä tähän on pluslasku 1 + 1: `(+ 1 1)`

```
_$ (+ 1 1)
2
```

Mikäli halutaan plussata useampi luku, kirjoitetaan ne kaikki komennon "+" jälkeen sulkujen sisään: 1 + 2 + 3 on lispissä `(+ 1 2 3)`

```
_$ (+ 1 2 3)
6
```

Jos haluamme yhdistää komentoja, jokaiselle komennolle on tehtävä omat sulut. Tämä voi aluksi hämätä laskutoimituksissa, mutta on ainakin helppo muistaa. Esimerkkinä lasku 10 / (1 + 1): `(/ 10 (+ 1 1))`

```
_$ (/ 10 (+ 1 1))
5
```

AutoCAD-komennot lispin kautta

AutoCAD-työskentelyssä tulee useasti eteen tilanteita, joissa syötetään monesti komentoja samoilla asetuksilla. Lisp tarjoaa mahdollisuuden tehdä omia komentoja, joilla voi yhdistellä olemassa olevat komennot paremmiksi omaan käyttöön. Esimerkiksi TEXT-komento kysyy tekstin korkeutta ja kulmaa, vaikka usein pitäisi vain saada teksti paikoilleen. Lisp-komennossa voidaan antaa nämä oletukset:

```
(command "TEXT" (getpoint) 5.0 0.0)
```

Tästä saa tehtyä oman komennon kirjoittamalla siitä funktiomäärittelyn. Laitetaan halutun toiminnon ympärille `defun` ja annetaan komennolle nimi `c:teksti`:

```
(defun c:teksti ()
  (command "TEXT" (getpoint) 5.0 0.0)
)
```

Tämän jälkeen AutoCAD ehdottaa komentoa "TEKSTI", joka viittaa yllä olevaan funktioon. `c:` laitetaan nimen eteen, että AutoCAD tunnistaa tämän olevan piirtotilassa käytettävä komento.

Käyttäjältä voi myös kysyä tiedot etukäteen, tallentaa ne muuttujaan, ja sitten käyttää niitä useassa komennossa. Alla esimerkkikomento "ALUE", joka kirjoittaa alueen nimen ja piirtää nelikulmion sen ympärille. Tekstin alkupistettä siirretään yksi oikealle ja alas, jotta se ei ole nelikulmion päällä.

```
(defun c:alue ()
  (setq teksti (getstring "Anna alueen nimi: "))
  (setq ekapiste (getpoint "Ensimmäinen piste: "))
  (setq tekstipiste (list (+ (car ekapiste) 1) (- (cadr ekapiste) 1)))
  (command "TEXT" "Justify" "TL" tekstipiste 5.0 0.0 teksti)
  (command "RECTANG" ekapiste)
  (princ)
)
```

Tallennus muuttujaan tapahtuu komennolla `setq`, jolle annetaan muuttujan nimi ja mitä tietoa tallennetaan - `(setq muuttuja tieto)`.

Getpoint, getstring, ...

Lispissä annetaan komennoille tietoa oikeassa tietotyypissä. Tämä onnistuu get-funktioilla.

- `getpoint` kysyy käyttäjältä pisteen. Palauttaa listan `'(x y z)`
- `getstring` kysyy tekstiä. Jos tekstiin haluaa välilyöntejä, kirjoitetaan teksti lainausmerkkeihin
- `getint` kysyy kokonaisluvun.
- `getreal` kysyy reaaliluvun.
- `getdist` kysyy kaksi pistettä ja palauttaa näiden etäisyyden
- `getfiled` kysyy tiedoston. Palauttaa tiedostopolun merkkijonona.

`getpoint`, `getstring`, `getint` ja `getreal` toimivat joko sellaisenaan tai viestin kanssa. Viesti ilmestyy AutoCADin komentoriville.

```
(getpoint)
(getpoint "Anna piste: ")
```

`getdist` sellaisenaan kysyy kaksi pistettä, mutta sille voi myös syöttää yhden pisteen koodissa, jolloin `getdist` kysyy vain yhden pisteen lisää. Viimeinen argumentti on viesti, kuten muissa get-funktioissa. Alla esimerkkejä, miten tätä funktiota voi kutsua:

```
(getdist)
(getdist "Pisteiden etäisyys, valitse piste:")
(getdist '(10 10))
(getdist '(10 10) "Pisteiden etäisyys, valitse piste:")
```

`getfiled` tarvitsee enemmän tietoja toimiakseen: otsikon, oletustiedostonimen, tiedostopäätteen/-päätteet ja asetukset.

```
(getfiled otsikko oletustiedostonimi tiedostopääte asetukset)
```

Asetuksissa on paljon vaihtoehtoja, jotka kannattaa katsoa AutoCADin dokumentaatiosta. Kaksi yleishyödyllisintä ovat 0 ja 1: 0 tarkoittaa olemassa olevan tiedoston hakua ja 1 uuden tiedoston luontia.

```
(getfiled "Valitse tiedosto: " "vanha.txt" "*" 0)
(getfiled "Anna tiedostopolku: " "uusi.txt" "*" 1)
```

Ohjelman kulun hallinta

Toisin kuin skripteissä, lispissä ohjelman kulkua voi ohjata päätösrakenteilla kuten `if` ja `cond`, ja toistorakenteilla kuten `while`, `repeat` ja `foreach`.

if

`if` päättää ehdon perusteella, kumpi kahdesta komennosta suoritetaan. Pseudokoodilla toimintaa voisi kuvata näin:

```
Jos ehto on tosi,
  suoritetaan komento yksi.
Muuten
  suoritetaan komento kaksi.
```

Lispissä tämä kirjoitetaan näin:

```
;Monella rivillä
(if ehto
  (komento-yksi)
  (komento-kaksi)
)

;Yhdellä rivillä
(if ehto (komento-yksi) (komento-kaksi))
```

if - Visual Lisp Console:

```
_$ (if (= 1 1) "eka eli tosi" "toka eli epätosi")
"eka eli tosi"
_$ (if (= 1 2) "eka eli tosi" "toka eli epätosi")
"toka eli epätosi"
```

Ehdot ja vertailu

AutoLISPissä vertailu tapahtuu komennoilla:

- `<` pienempi kuin
- `<=` pienempi tai yhtä suuri kuin
- `>` suurempi kuin
- `>=` suurempi tai yhtä suuri kuin
- `=` yhtä suuri kuin
- `/=` eri suuri kuin
- `equal` ovatko samanlaisia*
- `eq` ovatko sama*

Näitä käytetään kuten mitä tahansa muuta komentoa. Vertailu, joka on tosi, palauttaa `t`, ja epätosi palauttaa `nil`. Ehtolauseet voivat myös olla muuttujia sellaisenaan. Lisp tulkitsee kaiken, paitsi `nil`-arvon, todeksi.

Joskus tarvitaan, että useampi ehto toteutuu ennen kuin suoritetaan komento. Näissä tilanteissa voidaan käyttää komentoja `and` ja `or`. `and` palauttaa t vain, jos kaikki sen argumentit ovat tosia. `or` palauttaa tosi, jos yksikin sen argumenteista on tosi. Ehtojen arvoa voi myös kääntää `t -> nil` tai `nil -> t` komennolla `not`.

Ehdot ja vertailu - Visual Lisp Console:

```
_$ (< 1 2)
T
_$ (< 2 1)
nil
_$ (if "tekstiä" 'TOSI 'EPÄTOSI)
TOSI
_$ (if nil 'TOSI 'EPÄTOSI)
EPÄTOSI
_$ (and t t)
T
_$ (and t nil)
nil
_$ (or nil nil)
nil
_$ (or t nil)
T
_$ (not t)
nil
_$ (not nil)
T
```

*) `equal` vertaa, onko verrattavien muuttujien sisältö samanlainen, `eq` vertaa osoittavako verrattavat muuttujat samaan asiaan muistissa; `eq` voi palauttaa `nil` vaikka muuttujien sisältö olisi samanlainen, mutta ne sijaitsevat eri paikassa muistia.

progn

Komennolla `progn` voi ketjuttaa komentoja. Näin voi suorittaa useita komentoja kohdassa, jossa odotetaan vain yhtä komentoa. Tätä käytetään usein `if`:in kanssa:

progn - Visual Lisp Console:

```
_$ (if (< 1 2)
     (progn
       (princ "Tosi")
       (princ "\nProgn voi suorittaa monta komentoa")
       (princ)
     )
     (progn
       (princ "Epätosi")
       (princ)
     )
   )
Tosi
Progn voi suorittaa monta komentoa
```

cond

`cond` on kuin monta `if`-lausetta yhdessä. Se käy ehdon kerrallaan läpi, ja suorittaa ensimmäisen totta olevaa ehtoa vastaavat komennot. Viimeiseksi ehdoksi laitetaan yleensä aina tosi ehto sekä sen kanssa oletuskomento, joka suoritetaan jos mikään muu ehdoista ei ole toteutunut:

```
(cond
  (ehto-yksi
    (komento-yksi)
  )
  (ehto-kaksi
    (komento-kaksi)
  )
  (t
    (oletuskomento)
  )
)
```

while

`while` toistaa komentoja niin kauan kuin ehto on tosi. Ehto tarkastetaan, ja jos se on tosi, suoritetaan komennot `while`:n sulkujen sisällä. Näiden komentojen jälkeen tarkastetaan ehto uudelleen. Tätä jatketaan, kunnes ehto ei enää ole tosi. Mikäli ehto on aina tosi, ohjelma jatkaa silmukkaa loputtomiin. Tällaisista tilanteista selviää usein painamalla `esc`-näppäintä pohjassa.

```
(while ehto
  (komento-yksi)
  (komento-kaksi)
  (komento-kolme)
)
```

while - Visual Lisp Console:

```
_$ (setq i 0)
   (while (< i 10)
     (princ i)
     (setq i (1+ i))
     (princ)
   )
0
0123456789
```

repeat

`repeat` toistaa komentoja niin monta kertaa kuin käsketään.

```
(repeat montako-kertaa
  (komento-yksi)
  (komento-kaksi)
  (komento-kolme)
)
```

repeat - Visual Lisp Console:

```
_$ (repeat 5
     (princ "Kierros loopissa\n")´
     (princ)
   )
Kierros loopissa
Kierros loopissa
Kierros loopissa
Kierros loopissa
Kierros loopissa
```

foreach

`foreach` suorittaa komennot jokaista listan jäsentä kohden. Jäsen tallennetaan väliaikaisesti muuttujaan, jota voi käyttää komennoissa `foreach`:n sulkujen sisällä.

```
(foreach jäsen lista
  (komento-yksi)
  (komento-kaksi)
  (komento-kolme)
)
```

foreach - Visual Lisp Console:

```
_$ (setq lista (list "Tämä" "on" "lista" "tekstejä"))
   (foreach teksti lista
     (princ teksti)
     (princ "\n")
     (princ)
   )
("Tämä" "on" "lista" "tekstejä")
Tämä
on
lista
tekstejä
```