GUIDA PRATICA INTRODUTTIVA AD AutoLISP

INTRODUZIONE AD AUTOLISP

LA PROGRAMMAZIONE

Un **programma** (software) consiste essenzialmente in una sequenza di istruzioni scritte in un linguaggio comprensibile da parte del computer e strutturate in modo tale da far eseguire al computer determinate azioni

Con il termine **programmazione** si identificano tutte le operazioni che servono a trasformare in operativa l'idea di far fare una cena cosa al computer.

In genere si è portati a pensare che la programmazione non abbia nulla a che fare con la normale à di utilizzo del computer; si ritiene che essa interessi esclusivamente i programmatori sionisti, vale a dire coloro che Scrivono i programmi applicativi utilizzati nei vari settori d'impiego del computer. In base ad un simile presupposto l'unico compito di chi sfrutta il calcolatore per pria attività professionale sarebbe quello di imparare ad usare i programmi applicativi, adattando il proprio metodo di lavoro alla logica di funzionamento del software così come è stata impostata dai programmatori.

Anche se in effetti le cose stanno molto spesso cosi, vale la pena di fare una considerazione. Se è vero che la programmazione può essere sfruttata da tecnici esperti per concretizzare progetti software ambiziosi e complessi è altrettanto vero che si possono scrivere dei programmi più semplici, magari anche **molto semplici**, perfettamente adatti ad *automatizzare* e quindi *velocizzare sensibilmente*, particolari operazioni che si presentano con una certa frequenza nell'ambito di una attività lavorativa con l'aiuto del calcolatore. In questi casi risulta certamente più efficace ed economico, tutto per un **utilizzatore** di computer, essere autosufficiente.

Va detto anche che tuffi i prodotti software più rinomati ed importanti, proprio per venire incontro alle esigenze estremamente eterogenee degli utilizzatori, prevedono metodi più o meno sofisticati di personalizzazione delle funzioni. Alla base di una simile strategia c'è la constatazione del fatto che, anche nel caso di software progettati per un utilizzo specifico (ad esempio CAD topografico, elettrico, ecc.), non sarà mai possibile creare un programma che disponga di una gamma di funzionalità cosi ampia da soddisfare le particolari necessità di tutti potenziali utilizzatori.

Imparare a conoscere ed a sfruttare i sistemi di personalizzazione rappresenta una tappa fondamentale per qualunque utente di computer che voglia avere la padronanza dello strumento con cui opera.

Mi sembra controproducente l'atteggiamento di coloro che pur lavorando ogni giorno con AutoCAD o con altri sofisticati programmi, si ritengono estranei a questi argomenti in base alla convinzione che:

per quello che devo fare io, ne so abbastanza

E' indiscutibile che se utilizziamo tutti i giorni uno strumento di lavoro senza saperlo sfruttare pienamente avremo come risultato un notevole spreco di risorse, che sicuramente ci farebbe un certo effetto se venisse quantificato alla fine dell'anno in termini monetari. Questo non significa che si debba passare tutto il tempo a studiare; è sufficiente essere consapevoli del fatto che ogni piccolo investimento anche solo di tempo nella direzione di una maggiore padronanza degli strumenti software rappresenta un passo importante nella propria crescita professionale perché permette di concretizzare sempre meglio le proprie idee ed intuizioni.

LINGUAGGI DI PROGRAMMAZIONE

Il linguaggio che viene usato per scrivere un programma si dice **1inguaggio di programmazione** e consiste in un insieme di regole sintattiche e strutturali che identificano un certo modo di impartire istruzioni al calcolatore.

Dal punto di vista tecnico il calcolatore è in grado di comprendere un programma solamente se le istruzioni sono espresse nel cosiddetto **linguaggio macchina**, che descrive le azioni da svolgere mediante una successione di ordini elementari codificati in modo da essere direttamente comprensibili al elaboratore (codice binario); questo linguaggio viene detto di *basso livello* e può variare a seconda del tipo di computer.

Di fatto però i programmi vengono scritti usando dei linguaggi molto più evoluti, di *alto livello*, che sono stati sviluppati per permettere ai programmatori di scrivere i programmi in una forma più leggibile, utilizzando dei codici simbolici (anche alfanumerici) e delle strutture logiche sofisticate, in modo da ottenere delle sequenze di istruzioni la cui logica sia più vicina a quella umana che non a quella del calcolatore.

Ovviamente un programma scritto in linguaggio di alto livello, per poter essere eseguito da un certo elaboratore, dovrà prima in qualche modo essere tradotto nel linguaggio macchina dell'elaboratore stesso.

L'idea che sta alla base della nascita e dello sviluppo dei linguaggi di programmazione evoluti è quella di far eseguire allo stesso computer la traduzione delle istruzioni dal codice simbolico, detto anche **codice sorgente**, al codice macchina; allo scopo esistono dei programmi appositi, che in linea di massima si possono suddividere in due categorie in base al metodo seguito per effettuate la traduzione:

COMPILATORI	Traducono un intero programma sorgente creandone una versione in linguaggio macchina (file di tipo EXE, COM, ecc.). Un programma compilato risulta molto veloce nel caricamento e nell'esecuzione, in quanto la traduzione è già stata eseguita Per modificare il programma bisogna intervenire sul codice sorgente e poi effettuare di nuovo la compilazione. Questo spiega come mai anche un programmatore professionista non può fare delle modifiche a programmi acquistati (come AutoCAD, WORD, EXCEL, ecc,); il codice sorgente infatti, a meno di particolari accordi di collaborazione non viene mai messo a disposizione dalle aziende produttrici di software.
INTERPRETI	Traducono il programma istruzione per istruzione e causano l'immediata esecuzione delle istruzioni stesse. In questo caso non viene prodotto alcun file eseguibile, in quanto la traduzione avviene durante l'esecuzione del programma che risulterà di conseguenza più lenta. Le operazioni di modifica dei programmi sono più agevoli perché subito dopo aver corretto o cambiato qualche istruzione si può lanciare di nuovo l'esecuzione del programma

SCRIVERE UN PROGRAMMA

Ad eccezione di quanto accade con i linguaggi di programmazione espressamente studiati per l'interfaccia grafica di WIDOWS, un programma viene solitamente creato scrivendo in un file ASCII la sequenza delle istruzioni atte a svolgere il compito desiderato. Tale file può essere creato con un qualunque editor (come "Notepad") o mediante un programma di elaborazione testi in modalità non documento.

Il primo passo, ad ogni modo, consiste nello

stabilire chiaramente che cosa il programma deve fare,

individuando la **sequenza logica** delle operazioni. Ad esempio, se vogliamo creare un programma che chieda all'utente un valore e poi disegni automaticamente un cerchio di raggio pari al valore impostato, possiamo immaginare una sequenza di questo tipo:

1. apparizione sullo schermo di un messaggio di richiesta per il raggio del cerchio.
2. attesa di un valore da tastiera.
3. verifica della correttezza del valore ricevuto (l'operatore ha risposto in maniera accettabile?).
4. apparizione sullo schermo di un messaggio di richiesta per la posizione del centro dei cerchio.
5. attesa della indicazione di un punto da parte dell'utente.
6. disegno di un cerchio con centro nel punto specificato e di raggio pari al valore ricevuto al punto 2.

Nel gergo della programmazione la tabella qui sopra rappresenta il **diagramma di flusso** del programma che si sta progettando, Si tratta della struttura logica del programma, scritta in una forma del tutto *indipendente dal linguaggio* di programmazione che si utilizzerà. Il passo successivo è ovviamente quello di creare il file ASCII traducendo i

concetti in opportune istruzioni, che devono essere strutturate secondo le regole sintattiche previste dal linguaggio di programmazione utilizzato.

Uso delle variabili

II vantaggio più evidente offerto dall'utilizzo di un linguaggio di programmazione sta nella possibilità di creare delle **procedure interattive** i cui effetti cambiano di volta in volta sulla base di determinate scelte. Siamo quindi ad un livello nettamente superiore rispetto alla semplice automazione di sequenze di comandi, in cui l'utente non ha modo di intervenire durante l'esecuzione (si pensi ad esempio ai files di SCRIPT di AntoCAD).

Se si scrive uno script che disegna un rettangolo 50x30, ad ogni esecuzione dello script si otterrà sempre una figura di quelle dimensioni; preparando una procedura LISP, invece, si potrà fare in modo che il computer chieda di volta in volta all'utente le misure del rettangolo prima di disegnano. Oppure si potrebbe impostare il programma in modo che la macchina chieda determinati dati e poi calcoli automaticamente le misure del rettangolo sulla base dei dati inseriti e di una certa sequenza di calcoli da noi impostata (si pensi, ad esempio, alla necessità di. inserire la misura della base del rettangolo e l'area che esso deve avere, lasciando al programma il compito ci calcolare l'altezza e disegnare la figura).

Questo tipo di prestazioni è reso possibile dall'utilizzo, in fase di programmazione, delle cosiddette **variabili**. In maniera molto semplice, si possono immaginare le variabili come dei **segnaposto** che vengono usati per scrivere le istruzioni.

A titolo di esempio, prendiamo l'espressione

120 x 4

ogni volta che si esegue questo calcolo, il risultato è ovviamente 480.

Se l'espressione viene invece scritta nella forma

AxB

il risultato cambia di volta in volta a seconda dei valori attribuiti alle lettere A e B, che vengono proprio per questo chiamate *variabili*.

Possiamo concludere che, quando il computer chiede un valore, questo viene memorizzato dal programma in una variabile per poi essere utilizzato nei calcoli.

Le variabili possono quasi sempre avere un nome formato da più di un carattere (dipende dalle regole sintattiche del linguaggio utilizzato); questo rende più semplice la programmazione e più leggibili i programmi. Infatti, immaginando di dover calcolare il perimetro di un poligono regolare, l'espressione

LUNGH_LATO x NUM_LATI

è sicuramente più facilmente interpretabile rispetto a

LxN

Ovviamente la differenza non influisce in alcun modo sulla funzionalità di un programma, ma è determinante per la sua leggibilità e per rendere più veloci le fasi di correzione e/o modifica.

COLLAUDARE UN PROGRAMMA

Il **collaudo** e la **verifica** rappresentano una fase estremamente importante nell'ambito della programmazione. L'effettiva utilità di un programma è direttamente proporzionale alla sua **affidabilità**, prima ancora che alle sue caratteristiche potenziali. Per chiarire questo concetto basta pensare che nessuno di noi utilizzerebbe per il proprio lavoro un programma molto bello molto potente, molto veloce ma soggetto a saltuari ed inaspettati blocchi con conseguente perdita dei dati. Il rischio di perdere i dati sarebbe sufficiente a farci rinunciare alle pur interessanti potenzialità di quel programma.

Perché un programma sia affidabile deve essere privo di errori, o per lo meno privo di errori pericolosi dal punto di vista della salvaguardia del lavoro di chi dovrà usarlo. Una volta terminata la scrittura di un programma, pertanto, bisognerà prevedere sempre una accurata fase di collaudo durante la quale si dovranno creare tutte le possibili situazioni previste durante l'utilizzo del programma, comprese le **situazioni** cosiddette **non ovvie**: cosa succede se si risponde in maniera errata ad una domanda, oppure si assegna ad un certo parametro un valore non corretto, ecc.?

Il collaudo dei propri programmi diventa essenziale nel caso in cui essi debbano essere utilizzati da terze persone, che come e ovvio non sono al corrente della logica che si è seguita nella stesura del programma e che pertanto potrebbero facilmente dare luogo a situazioni non previste.

Questo spiega come mai molte aziende produttrici di software coinvolgono direttamente alcuni clienti (noti come beta-testers nelle operazioni di collaudo e messa a punto che precedono, anche di molti mesi, il lancio sul mercato di un nuovo programma.

IL LINGUAGGIO AUTOLISP

AutoLISP è una versione particolare del linguaggio di programmazione ***LISP*** (LISt Processing, elaborazione di liste) integrata nel software AutoCAD e dotata di funzionalità particolari di interazione con AutoCAD stesso.

AutoLISP rappresenta uno dei più potenti strumenti di **personalizzazione** offerti da AutoCAD; mediante AutoLISP si può intervenire sul funzionamento di AutoCAD in modo anche molto complesso ed efficace. Si va dalla semplice **automazione di operazioni** frequenti alla scrittura di **macroistruzioni** od anche di veri e propri **programmi** con cui rendere l'ambiente di lavoro il più possibile adatto alle proprie esigenze, aggiungendo per esempio dei comandi specifici che permettano la soluzione di particolari problemi ricorrenti.

STRUTTURA DEL LINGUAGGIO

Nel linguaggio LISP il ruolo fondamentale è ricoperto dalle **lista** (dette anche *elenchi*).

Una lista è sequenza di entità, che possono essere caratteri, numeri, caratteri speciali nomi, parole chiave, altre liste, ecc. Le singole entità, che vengono dette **elementi**, sono separate fra di loro da almeno uno spazio. L'intera lista è delimitata da parentesi rotonde. Per esempio:

(A B C D) e un elenco con quattro membri

(1 7 3 456.67 Giovanni) è un elenco con cinque membri

La tabella seguente illustra i principali tipi di dati che AutoLlSP utilizza; la loro conoscenza è fondamentale per comprendere il modo in cui le informazioni vengono elaborate.

NUMERO INTERO	numero che non presenta cifre decimali. Anche se AutoLISP utilizza internamente numeri interi a 32 bit (compresi fra -2^{31} e $+2^{31}$), nei sistemi DOS i valori trasferiti fra AutoLISP ed AutoCAD sono limitati ai 16 bit, per cui un numero intero immesso in AntoCAD può andare da –32768 (-2^{15}) a +32768 ($+2^{15}$).
NUMERO REALE	numero che presenta cifre decimali. Un numero reale va sempre indicato con il punto decimale e con almeno una cifra dopo di esso. I numeri compresi fra -l e +1 devono avere uno zero prima del punto decimale. AutoLISP memorizza ed elabora i numeri reali nel formato a virgola mobile a doppia precisione, che garantisce una accuratezza dei calcoli di almeno 14 cifre significative, anche se nell'area della riga di comando di AutoCAD ne vengono visualizzate meno. I numeri reali possono anche essere espressi in notazione scientifica; ad esempio, il numero 0.000082 equivale a 8.2^{E-5}. E' importante notare che, agli occhi di AutoLISP, la differenza tra un numero reale ed uno intero sta nella presenza o meno del punto decimale, non del fatto che dopo di esso ci siano degli zeri o altri numeri. Pertanto, ad esempio, il numero 5.0 viene considerato come numero reale.
PUNTO 2D	lista di due numeri reali, intesi come coordinate X ed Y di un punto. Ad esempio: **(12.56 4.0)**
PUNTO 3D	lista di tre numeri reali, intesi come coordinate x, y e z di un punto Ad esempio: **(-23.123 0.8 5.5)**
SIMBOLO (variabile)	elemento di lista che non è un puro numero, e che rappresenta, a seconda dei casi, il nome di una variabile o di una funzione. In particolare, il nome di una variabile può essere formato da una sequenza qualunque di caratteri, maiuscoli o minuscoli, il primo dei quali non deve essere una cifra; non sono permessi i caratteri: **( ) ‘ ” ;** Nell'assegnare i nomi alle variabili bisogna evitare quei nomi che sono già usati da AutoLISP. Per ottenere una lista dei nomi di simbolo da evitare si usa la funzione **ATOMS-FAMILY.** Per assegnare un valore ad una variabile si utilizza la funzione **SETQ**. Ogni variabile conserva l'ultimo valore assegnatole, fino a quando si esce dal disegno corrente.

STRINGA	sequenza di caratteri delimitata da virgolette. Il numero di caratteri di una singola stringa può arrivare al massimo a 132 (la memoria viene assegnata dinamicamente); si possono definire stringhe di lunghezza superiore mediante la funzione **STRCAT**. Nelle stringhe il carattere barra rovesciata assume un significato particolare in quanto viene usato per includere dei caratteri di controllo, alcuni dei quali sono: \\ → carattere \ \" → carattere " \n → a capo \r → invio

Tutte le espressioni AutoLISP si riconducono allo schema seguente:

(***nome_funzione*** [argomento]…[argomento]…)

Ogni espressione inizia con una parentesi rotonda aperta ed è costituita da un nome di funzione e da un elenco opzionale di argomenti, ciascuno dei quali può essere esso stesso un'espressione. L'espressione termina con una parentesi rotonda chiusa.

La quantità e natura degli argomenti varia in base alla funzione.

LA VALUTAZIONE DELLE ESPRESSIONI

La totale integrazione fra AutoCAD ed AutoLISP è ottenuta grazie alla costante presenza in memoria, durante le sessioni di lavoro AutoCAD, dell'**interprete AutoLISP**.

Quando in risposta al messaggio "*Comando:*" di AutoCAD si scrive un'espressione AutoLISP, questa viene subito riconosciuta dall'interprete, che la elabora mediante un apposito **programma di valutazione** e ne restituisce il risultato. Le espressioni nidificate all'interno di altre espressioni restituiscono i loro risultati alle espressioni che le includono.

A titolo di esempio si consideri l'espressione:

(+ 12 3 (* 5 4))

In questa espressione la funzione di somma (+) ha 3 argomenti, il terzo dei quali è a sua volta un'espressione, la funzione di moltiplicazione (*). Il risultato della moltiplicazione viene usato come argomento per la somma, ottenendo così il numero 35 (12+3+20).

Dal momento che l'interprete AutoLISP è sempre attivo durante le sessioni AutoCAD risulta possibile usare espressioni AutoLISP anche in risposta a numerosi messaggi di richiesta lanciati dai comandi standard di AutoCAD. L'importante è che si utilizzi, di volta in volta, un'espressione che restituisca un risultato compatibile con il messaggio di richiesta; ad esempio, se quando il comando OFFSET chiede la distanza di sfalsamento viene inserita l'espressione *(/ 60 12)* si otterrà come distanza di offset il valore 5, risultato della divisione 60/12.

Si può dare in risposta ad un messaggio AutoCAD il valore di una variabile AutoLISP scrivendo il nome della variabile preceduto da un punto esclamativo. Si supponga ad esempio di aver assegnato il valore 12 alla variabile raggio mediante l'espressione:

(setq raggio 12)

Quando AutoCAD presenta un messaggio di richiesta relativo ad una distanza si può rispondere con **!raggio**; il valore della variabile (in questo caso 12) verrà interpretato come risposta. Questo meccanismo può essere ad esempio sfruttato per memorizzare e richiamare velocemente numeri e/o nomi difficili da ricordare oppure lunghi da scrivere

---

STESURA DI UN PROGRAMMA

Sebbene le espressioni AutoLISP possano essere scritte direttamente in AutoCAD in risposta al messaggio "*Comando:*" il metodo più opportuno per sfruttare le potenzialità di questo strumento di personalizzazione consiste nel creare dei programmi che possano essere richiamati ed eseguiti oppure analizzati ed aggiornati in qualunque momento

Un programma AutoLISP deve essere memorizzato in un file ASCII al quale si assegna solitamente l'estensione LSP o MNL. Il programma consiste in una serie di espressioni che dovranno essere valutate al momento dell'esecuzione. Le espressioni vengono solitamente raggruppate (mediante la funzione ***DEFUN***) in una o più funzioni definite dal programmatore in base alla logica con cui il programma viene strutturato.

Nella stesura del file bisogna fare attenzione a bilanciare sempre correttamente le parentesi aperte con quelle chiuse, dal momento che molto frequentemente le liste vengono **annidate** (una lista ne contiene altre, che a loro volta contengono altre liste ancora, ecc.). Se qualche parentesi è messa nel posto sbagliato o manca addirittura può risultare difficile localizzare l'errore. fortunatamente in questo linguaggio di programmazione gli spazi e le eventuali andate a capo sono ininfluenti, per cui è possibile rendere più chiara la struttura delle espressioni saltando o rientrando delle righe.

Una volta che il programma è stato salvato bisogna entrare in AutoCAD, caricare in memoria il file LSP mediante la funzione ***LOAD*** e poi lanciare l'esecuzione vera e propria.

Se dopo il lancio del programma si ottengono dei messaggi di errore e/o dei malfunzionamenti (nel senso che non succede ciò che ci si aspetta) molto probabilmente ci sono degli errori di sintassi o di struttura nel file ASCII. In questi casi, ed in generale ogni qualvolta si desideri modificare un programma, bisognerà intervenire sul file ASCII corrispondente, ricaricarlo e lanciare di nuovo l'esecuzione.

Un aspetto di grande importanza riguarda l'inserimento, nel file ASCII, di annotazioni (commenti) che spieghino in modo essenziale ma chiaro, la logica di funzionamento del programma stesso. Lo scopo è quello di rendere il più semplici possibile gli interventi di correzione e/o modifica che il programma dovrà subire in futuro; infatti, dopo un po' di tempo, si tende a non ricordare più con esattezza il ragionamento fatto per la stesura di un

programma e quindi, per poterlo modificare in modo efficace, è molto importante trovare nel programma stesso delle annotazioni esplicative.

In ogni riga di programma AutoLISP è possibile inserire dei commenti, purché essi siano preceduti da un punto e virgola; tutti i caratteri dal punto e virgola fino alla seguente andata a capo verranno ignorati in fase di esecuzione del programma.

FUNZIONI PRINCIPALI DI AUTOLISP

La funzione di assegnazione

La funzione **SETQ**, detta *funzione di assegnazione*, viene usata per assegnare alle variabili i valori che esse devono memorizzare. Ogni volta che, ad esempio, si desidera far apparire un messaggio che richieda all'utente un certo dato, oltre alla funzione specifica relativa al dato da richiedere (vedi funzioni di tipo *GETxxx*) si dovrà invocare la funzione di assegnazione affinché il dato inserito dall'utente sia memorizzato in una variabile e sia quindi utilizzabile in seguito nell'ambito di altre espressioni LISP presenti nel programma.

(**setq** variabile1 valore1 [variabile2 valore2]…)	Assegna alla *variabile1* il *valore 1*, alla *variabile2* il *vaolore2* e cosi via. La funzione restituisce sempre l'ultimo valore assegnato)	
	(setq raggio 25.4)	assegna alla variatile *raggio* il valore 25.4; d'ora in avanti, ogni volta che la variabile *raggio* sarà valutata essa restituirà tale valore, fino a che non le sarà assegnato un valore diverso tramite un'altra espressione **SETQ**.
	(setq raggio **(getreal "Raggio?")** **)**	assegna alla variabile *raggio* il numero reale restituito dalla funzione GETREAL (vedi oltre), che a sua volta lo riceve da un input dell'utente.

Funzioni matematiche

(+ [numero numero...])	restituisce la somma di tutti i numeri indicati; se anche uno solo degli addendi è un numero reale, il risultato sarà un numero reale. Nel caso in cui non vi siano argomenti la funzione restituisce zero. **(+ 1 2)** ⇒ 3 **(+ 3 5 5.0)** ⇒ 12.0
(**-** [numero numero...])	sottrae il secondo numero dal primo e restituisce il risultato, se sono dati più di due numeri la somma di tutti meno il primo viene sottratta dal primo; se è dato un solo numero, esso viene sottratto da zero. Nel caso in cui non vi siano argomenti la funzione restituisce zero. **(- 50 40)** ⇒ 10 **(- 50 40 2)** ⇒ 8 **(- 10)** ⇒ -10

(***** [numero numero...])	restituisce il prodotto di tutti i numeri; se è dato un solo numero, viene moltiplicato per l'unità. Nel caso in cui non vi siano argomenti la funzione restituisce zero. **(* 4 7.0 2)** ⇒ 56.0
(/ [numero numero...])	restituisce il risultato della divisione del primo numero per il secondo; se sono dati più di due numeri, il primo viene diviso per il prodotto di tutti gli altri; se è dato un solo numero, viene diviso per l'unità. Nel caso in cui non vi siano argomenti la funzione restituisce zero **(/ 14 7.0)** ⇒ 2.0
(**1 +** numero)	restituisce il numero, che può essere reale o intero, aumentato di uno. **(1 + -18.9)** ⇒ -17.9
(**1 -** numero)	restituisce il numero, che può essere reale o intero, diminuito di uno. **(1 - 89)** ⇒ 88
(**abs** numero)	restituisce il valore assoluto del numero, che può essere reale o intero. **(abs –34.567)**⇒ 34.567
(**atan** num1 [num2])	restituisce l'arcotangente di *num1*, espressa in radianti. Se è presente anche *num2*, si ottiene, in radianti, l'arcotangente di *num1/num2*. I valori restituiti da questa funzione sono sempre compresi fra e $-\frac{\pi}{2} e \frac{\pi}{2}$
(**cos** angolo)	restituisce, come numero reale, il coseno dell'angolo. L'angolo deve essere espresso in radianti.
(**fix** numero)	restituisce la conversione di numero (può essere intero o reale) in un intero ottenuto eliminando le eventuali cifre decimali. **(fix 12)** ⇒ 12 **(fix 36.78)** ⇒ 36
(**float** numero)	restituisce la conversione di numero (intero o reale) in un numero reale **(float 12)** ⇒ 12.0 **(fload 36.78)** ⇒ 36.78
(**gcd** intero1 intero2)	restituisce il massimo comune denominatore dei due interi specificati.
(**max** numero numero…)	restituisce il maggiore fra i numeri dati, che possono essere reali o interi; il risultato è un numero intero solo se tutti i numeri dati sono interi.
(**min** numero numero…)	restituisce il minore fra i numeri dati, che possono essere reali o interi; il risultato è un numero intero solo se tutti i numeri dati sono interi.
pi	Non è una funzione, ma la costante π (il valore è approssimato a 3.141596).
(**rem** num1 num2…)	divide il primo numero per il secondo e restituisce il resto della divisione. I numeri possono essere interi o reali. Se vengono forniti più di due numeri, la funzione divide il primo per il secondo; il risultato 4ella divisione viene poi diviso per il terzo numero, e così via.

(**sin** angolo)	restituisce, come numero reale, il seno dell'angolo. L'angolo deve essere espresso in radianti.
(**sqrt** numero)	restituisce, come numero reale, la radice quadrata del numero indicato

Funzioni di confronto e condizionali

L'utilizzo delle funzioni di confronto e delle istruzioni condizionali rappresenta uno degli strumenti più potenti della programmazione dei computer. Tramite queste strutture logiche, infatti, si possono creare programmi che si comportano in modo diverso in base al verificarsi o meno di determinate condizioni impostate dal programmatore. Le funzioni di confronto hanno l'unico scopo di stabilire se una certa condizione è vera (**T, true**) o falsa (**nil**). Impostando opportunamente le condizioni da verificare e le istruzioni da eseguire di conseguenza. si possono ottenere, da parte dei programmi, comportamenti molto efficaci.

(= elemento [elemento]…)	restituisce T se tutti gli elementi, numeri e/o stringhe, sono uguali come valore; in caso contrario restituisce **nil**. **(= 5 5.0)** ⇒ **T** **(= 20 12)** ⇒ **nil** **(= "ciao" "ciao")** ⇒ **nil**
(/= elemento elemento)	restituisce T se i due elementi, numeri e/o stringhe, sono diversi. **(/= 10 10)** ⇒ **nil**
(< elemento elemento…)	restituisce T se il primo elemento è minore del secondo; se sono dati più di due elementi, si ottiene T se ogni elemento è minore di quello alla sua destra. **(< 2 4 56)** ⇒ **T** **(< 2 4 4)** ⇒ **nil**
(<= elemento elemento…)	restituisce T se il primo elemento è minore o uguale al secondo; se sono dati più di due elementi, si ottiene T se ogni elemento è minore o uguale a quello alla sua destra.
(> elemento elemento…)	restituisce T se il primo elemento è maggiore del secondo; se sono dati più di due elementi, si ottiene T se ogni elemento è maggiore di quello alla sua destra.
(>= elemento elemento…)	restituisce T se il primo elemento è maggiore o uguale al secondo; se sono dati più di due elementi, si ottiene T se ogni elemento è maggiore o uguale rispetto a quello alla sua destra.
(**and** espressione…)	restituisce T se tutte le espressioni indicate producono T. se a=45, b=nil, c="ciao", allora **(and 12 a c)** ⇒ **T** (and 12 a b c) ⇒ **nil**

(cond (test1 risultato1)…**)**	accetta come argomenti un numero qualunque di liste; valuta il primo argomento di ogni lista finché uno di questi elementi restituisce un valore diverso da **nil**. A questo punto la funzione valuta le espressioni che seguono il *test* affermativo e restituisce il valore dell'ultima di tali espressioni. Di seguito è riportato un semplice esempio: supponendo che la variabile *risposta* contenga la stringa che l'utente ha battuto in risposta ad un messaggio che chiedeva *"Sì o no?"*, questa funzione verifica la risposta e restituisce **1** se tale risposta è **Y** oppure **y**, **0** se la risposta **N** oppure **n**; negli altri casi, restituisce **nil**. (cond ((= risposta "Y") 1) ((= risposta "y") 1) ((= risposta "N") 0) ((= risposta "n") 0) (t nil))
(**eq** espr1 espr2)	restituisce T se le due espressioni sono **identiche**, cioè se fanno riferimento alla stessa locazione di memoria. Due variabili cui è attribuito lo stesso valore, mediante due istruzioni di assegnamento analoghe (a = 12.5, b = 12.5), non risultano identiche agli occhi di questa funzione in quanto i due dati si trovano in locazioni di memoria diverse. Se ad una variabile viene invece assegnato il valore di un'altra (a = 12.5, b = a), le due risultano identiche.
(equal espr1 espr2 [approx]**)**	restituisce T se le due espressioni sono **identiche** come valore, indipendentemente dalla locazione di memoria. Se due liste sono considerate identiche da **equal** possono non esserlo per **eq**, mentre elementi considerati **equal** sono sempre anche **eq**. Espressioni considerate **eq** sono sempre anche **equal**. *Approx* specifica la differenza massima (tolleranza) che può intercorrere fra i valori delle due espressioni senza che esse siano considerate non **equal**. se a = 4.123456 e b = 4.123457 (**equal** a b) ⇒ **nil** (**equal** a b 0.00001) ⇒ **T**
(**if** espr_test espressione_allora [espressione_altrimenti])	valuta le espressioni in modo condizionale; se *espressione_test* è diversa da **nil** viene valutata l'*espressione_allora.* In caso contrario, se è presente l'*espressione_altrimenti*, essa viene valutata, altrimenti si passa alla funzione successiva nel programma La funzione restituisce sempre il valore dell'espressione valutata. Di seguito è riportato un esempio: supponendo che la variabile lati contenga il numero dei lati di un poligono da disegnare, questa funzione fa apparire sullo schermo una frase che dichiara il tipo di figura che sarà disegnata. (if (= lati 3) (princ "Verra disegnato un triangolo") (princ "Non verrà disegnato un triangolo"))
(**minusp** elemento)	restituisce **T** se l'elemento è un numero negativo (reale o intero); in caso contrario restituisce **nil**.

(**not** elemento)	restituisce **T** se l'espressione dà **nil**; in caso contrario restituisce **nil**. Se a = 123 e b = nil (**not** a) ⇒ **nil** (**not** b) ⇒ **T**
(**numberp** elemento)	restituisce **T** se l'elemento è un numero (reale o intero), altrimenti restituisce **nil**.
(**or** espress…)	restituisce l'OR logico di una lista di espressioni. Le espressioni vengono valutate da sinistra a destra e, non appena una di esse dà come risultato un valore diverso da **nil**, la valutazione si interrompe e la funzione restituisce **T**. Se tutte le espressioni valgono **nil**, la funzione restituisce **nil**.
(**repeat** numero espress…)	valuta sequenzialmente le espressioni tante volte quante ne indica il numero (che deve essere intero e positivo). Restituisce il valore dell'ultima espressione.
(**while** espr_test espress…)	valuta l'espr_test; se questa è diversa da **nil**, valuta le altre espressioni e poi rivaluta l'espr_test; se questa è ancora diversa da **nil**, valuta nuovamente le altre espressionie cosi via. Quando l'espr_test dà **T** la funzione termina e restituisce il valore dell'ultima espressione valutata. Ovviamente l'uso di questa funzione condizionale prevede che la valutazione delle cspressioni causi una variazione nell'esito dell'espressione test, altrimenti la valutazione delle espressioni verrebbe ripetuta all'infinito.
(**zerop** elemento)	restituisce **T** se l'elemento indicato è un numero, reale o intero, che vale zero; in caso contrario restituisce **nil**.

Funzioni di manipolazione delle liste

(**append** lista…)	restituisce un'unica lista che comprende tutte le liste specificate. (**append** '(a b) '(c d)) ⇒ (A B C D)
(**car** lista)	restituisce il primo elemento della lista specificata (**nil** se la lista è vuota) (**car** '(a b c)) ⇒ A
(**cdr** lista)	restituisce una lista che contiene tutti gli elementi di quella specificata, meno il primo. Se la lista data è vuota, restituisce **nil**. (**cdr** '(a b c)) ⇒ (B C)

(**caar** lista), (**cadr** lista), (**cddr** lista), (**cadar** lista), …	concatenazioni di **CAR** e **CDR**; sono previste concatenazioni fino a quattro livelli. Nei nomi di queste funzioni ogni **a** rappresenta una chiamata a **CAR**, mentre ogni **d** rappresenta una chiamata a **CDR**. Per esempio: **(caar x)** equivale a **(car (car x))** **(cdar x)** equivale a **(cdr (car x))** **(cadar x)** equivale a **(car (cdr (car x)))** **(cadr x)** equivale a **(car (cdr x))** **(cddr x)** equivale a **(cdr (cdr x))** **(caddr x)** equivale a **(car (cdr (cdr x)))** La Funzione **CADR** viene di solito usata per ottenere la coordinata Y di un punto 2D o 3D, vale a dire il secondo elemento di una lista di due o tre valori reali: **(cadr '(a b c))** ⇒ B Con **CADDR** si ottiene invece la coordinata Z (di un punto 3D).
(**cons** primo_elem lista)	restituisce una lista che è ottenuta aggiungendo *primo_elem* all'inizio della lista specificata; questa funzione si usa per la costruzione di liste.
(**last** lista)	restituisce l'ultimo elemento della lista indicata, la quale non deve essere vuota (**nil**) **(last '(a b c d))** ⇒ D **(last '(a b c (r t y)))** ⇒ (R T Y) Nel calcolo delle coordinate dei punti la funzione **LAST** può sembrare analoga a **CADR** per ottenere la coordinata Y. Questo è vero però solo per i punti 2D, dal momento che per i punti 3D **LAST** restituisce la coordinata Z. Al fine di evitare espressioni ambigue è bene pertanto preferire le funzioni **CADR** c **CADDR.**
(**lenght** lista)	restituisce un numero intero che indica il numero di elementi della lista indicata.
(**list** espr…)	restituisce una lista i cui elementi sono tutti gli argomenti specificati. **(list 'a 'b 'c)** ⇒ (A B C) **(list 'a '(b c) 'd)** ⇒ (A (B C) D) Questa funzione é usata spesso per definire una variabile di punto 2D o 3D (lista di due o tre numeri reali: **(setq punto1 (list 12 0.0 47))** Nei casi in cui nella lista non esistano variabili o elementi non definiti è possibile utilizzare la funzione **QUOTE** alternativa a **LIST**.
(**nth** n lista)	restituisce l'ennesimo elemento della lista indicata (n = 0 corrisponde al primo elemento); nel caso in cui *n* sia superiore al numero degli elementi di *lista*, la funzione restituisce **nil**.

(**quote** espress) oppure **'espress.**	non valuta l'espressione e la restituisce letteralmente. **(quote (12 0.0 47))** oppure **'(12 0.0 47)** ⇒ (12 0.0 47) **(quote a)** oppure **'a** ⇒ A La sintassi **'espress** non può essere usata come risposta da tastiera ad un messaggio di richiesta di AutoCAD. Questa funzione è usata spesso in alternativa a **LIST** per definire una variabile di punto 2D o 3D (lista di due o tre numeri reali): **(setq punto1 '(12 0.0 47))**
(**riverse** lista)	restituisce una lista che presenta gli stessi elementi della lista data, ma in ordine invertito.

Funzioni relative alle stringhe di testo

(**angtos** angolo [modo [precisione]])	restituisce una stringa ottenuta dalla conversione dell'angolo, specificato in radianti e come numero reale, Il formato della stringa dipende dagli argomenti *modo* e *precisione*, nonché dalle impostazioni di alcune variabili di sistema AutoCAD. Se *modo* = 0 l'angolo viene convertito in gradi decimali, mentre se il valore è 3, si ha l'espressione in radianti. Il parametro *precisione* è un intero che stabilisce il numero di cifre dopo la virgola. **(angtos 3.14159 0 2)** ⇒ 180.00 oppure 180 Sono stati previsti due possibili risultati in quanto **ANGTOS** si adegua alla impostazione della variabile di quotatura. DIMZIN (soppressione degli zeri ridondanti).
(**ascii** stringa)	restituisce il codice **ASCII** (numero intero) del primo carattere della stringa data.
(**atof** stringa)	restituisce un numero reale ottenuto dalla conversione della stringa specificata. **(atof "97.12")** ⇒ 97.12
(**atoi** stringa)	restituisce un numero intero ottenuto dalla conversione della stringa specificata. **(atoi "97.82")** ⇒ 97
(**chr** intero)	restituisce una stringa di un solo carattere ottenuta dalla conversione del numero intero specificato. Il codice ASCII del carattere corrispondente all'intero dato.
(**distof** stringa [modo])	restituisce un numero reale ottenuto dalla conversione della stringa indicata. L'argomento *modo* stabilisce in quale formato debba essere interpretata la stringa. Se *modo* = 2 la stringa viene interpretata come un numero in formato decimale.
(**itoa** intero)	restituisce una stringa ottenuta dalla conversione di un numero intero specificato. **(itoa –36)** ⇒ "-36"

(**rtos** numero [modo [precisione]])	restituisce una stringa ottenuta dalla conversione del *numero* reale indicato. Il formato della stringa dipende dagli argomenti *modo* e *precisione*, nonché dalle impostazioni di alcune variabili di sistema AutoCAD. Se *modo* = 2 il numero viene convertito in formato decimale. Il parametro *precisione* è un intero che stabilisce il numero di cifre dopo la virgola. Così come **ANGTOS** anche **RTOS** si adegua alla impostazione della variabile di quotatura DIMZIN (soppressione degli zeri ridondanti)
(**strcase** stringa [minusc_maiusc])	valuta la stringa indicata e restituisce una copia della stessa con tutti i caratteri alfabetici convertiti in maiuscole o minuscole, a seconda della presenza e del valore dell'argomento *minusc_maiusc.* *minusc_maiusc* assente => caratteri convertiti in maiuscole *minusc_maiusc* presente e = nil => caratteri convertiti in maiuscole *minusc_maiusc* presente e /= nil => caratteri convertiti in minuscole
(**strcat** stringa1 [stringa2]…)	restituisce una stringa che è la concatenazione delle stringhe indicate. (strcat "saltim" "banco" " svedese") => "saltimbanco svedese"
(**strlen** [stringa])	restituisce numero intero che rappresenta il numero di caratteri che compongono la stringa indicata. Se vengono date più stringhe in sequenza, si ottiene il numero complessivo di caratteri. Se si omette l'argomento, la funzione restituisce il numero intero zero.
(**substr** stringa inizio [lunghezza])	restituisce una stringa che corrisponde ad una parte della stringa specificata. L'argomento *inizio*, numero intero positivo, indica la posizione del carattere dal quale comincia la parte di stringa da estrarre (*inizio* = 1 corrisponde al primo carattere). L'argomento *lunghezza*, anch'esso intero positivo, stabilisce quanti caratteri debbano essere estratti. Se *lunghezza* viene omesso vengono estratti tutti i caratteri fino alla fine della siringa. **(substr "sfocato" 2 4)** ⇒ "foca" **(substr "cantante" 4)** ⇒ "tante"

Funzioni specifiche per l'ambiente CAD

AutoLISP mette a disposizione numerose funzioni specifiche per interagire con l'ambiente grafico di AutoCAD.

Un insieme di queste funzioni, accomunate dal prefisso "**get**", rivestono un ruolo fondamentale nella realizzazione di procedure interattive per AutoCAD) Tramite tali funzioni, infatti, un programma AutoLISP può richiedere dei dati di varia natura direttamente all'utente clic sta usando AutoCAD. Quando incontra una funzione **GETXXX**, l'interprete LISP arresta momentaneamente l'esecuzione del programma, scrive a video un messaggio di richiesta ed attende che l'utente immetta un certo dato. Se il dato ricevuto soddisfa le condizioni previste, l'esecuzione prosegue. I messaggi di richiesta sono impostabili a piacere, rendendo così possibile la perfetta integrazione delle procedure LISP nell' ambito dei normali comandi AutoCAD.

Nel seguito sono indicate le principali funzioni di interazione con AutoCAD.

(**angle** pt1 pt2)	restituisce, espresso in radianti, l'angolo che una retta passante per i due punti indicati forma con il semiasse positivo delle X dell'UCS corrente. **(angle '(5.0 3.5) '(10.4 10.6))** ⇒ 0.433581 Nel caso in cui i punti indicati siano di tipo 3D il calcolo viene fatta proiettandoli sul piano XY dell'UCS corrente.

(**command** [argomenti]…)	esegue comandi AutoCAD e restituisce sempre **nil**. Mediante questa funzione è possibile, da un programma LISP, lanciare qualunque comando AutoCAD; **COMMAND** permette anche di specificare le singole risposte da dare ai messaggi del particolare comando lanciato, di modo che si possono usare le variabili definite nel programma come valori da passare ai comandi AutoCAD. In questo modo ogni volta che il comando viene lanciato il programma potrà rispondere alle varie richieste in modi diversi, ottenendo risultati differenti. Gli *argomenti* sono i nomi dei comandi e dei relativi sottocomandi, nonché le risposte da dare ai vari messaggi di richiesta che ogni comando AutoCAD prevede. Ogni argomento viene valutato e inviato sequenzialmente all'editor grafico (come se venisse battuto sulla tastiera). I nomi dei comandi e dei sottocomandi sono da considerarsi stringhe, mentre gli altri argomenti possono essere stringhe, numeri reali, numeri interi o punti a seconda del comando richiamato. Una stringa nulla ("") equivale alla pressione del tasto ↵**,** mentre se la funzione **COMMAND** viene scritta senza argomenti produce l'effetto della combinazione di tasti ^C. Se la variabile di sistema CMDECHO ha valore zero i comandi richiamati non vengono visualizzati sullo schermo; questo accorgimento risulta utile quando, una volta verificato il corretto funzionamento di un programma, non si desidera la visualizzazione esplicita dei vari comandi AutoCAD che esso esegue. **(command "cerchio" "0,0" "d" "12")** ⇒ disegno di un cerchio in 0,0 con diametro 12 **(command "cerchio" "0,0" diam ragg)** ⇒ disegno di un cerchio in 0,0 con diametro e raggio pari ai valori correnti delle variabili *diam* e *ragg.* **(command "linea" punto1 punto2 "")** ⇒ disegno di un segmento che ha come estremi i punti le cui coordinate sono memorizzate nelle due variabili *punto1* e *punto2*. La stringa vuota finale serve a terminare il comando LINEA. Nell'ambito della funzione **COMMAND** non si possono utilizzare le funzioni di tipo **GETxxx.** Ogni tentativo in questo senso produce in rnessaggio di errore. Non si possono lanciare con questa funzione quei comandi AutoCAD che prevedono obbligatoriamente l'uso del mouse, come ad esempio TESTODIN, DDLMODI, DDINSERT. In questi casi si fa ricorso ai comandi come TESTO, LAYER e INSER, che permettono di ottenere i medesimi risultati con il solo input da tastiera. Si può indurre la funzione **COMMAND** a fermarsi durante l'esecuzione per permettere all'utente una immissione diretta; questo comportamento si ottiene inserendo fra gli argomenti. della funzione la speciale variabile predefinita **pause**. Nell'esempio **(command "inser" "tavolo" pause "" "" pause)** la funzione lancia l'inserimento nel disegno del blocco *tavolo*, sì ferma sulla richiesta del punto di inserimento, risponde con ↵ alle due richieste successive (fattori di scala) e poi si ferma nuovamente sulla richiesta dell'angolo di rotazione; le fermate durano fino a che l'utente non risponde. ai rispettivi messaggi.

(**distance** pt1 pt2)	restituisce la distanza fra i due punti indicati; se anche uno solo dei due punti è del tipo 2D le eventuali coordinate Z degli altri punti sono ignorate e la distanza viene misurata sul piano XY corrente con riferimento alle proiezioni dei punti sullo stesso.
(**getangl**e [punto] [richiesta])	introduce una pausa perché l'utente immetta (da tastiera o in modo dinamico) un angolo, il quale poi viene restituito in radianti e relativo al piano XY corrente ed alla impostazione della variabile ANGBASE. L'argomento *punto* è un punto 2D opzionale; se è presente attiva l'inserimento dell'angolo in modo dinamico e viene interpretato come primo punto della linea elastica disegnata da AutoCAD sullo schermo. L'argomento *richiesta* è una stringa opzionale; se è presente viene visualizzata sotto forma di messaggio di richiesta per l'utente
(**getcorner** punto [richiesta])	introduce una pausa perché l'utente immetta in modo dinamico un punto, che poi viene restituito come lista di due o tre numeri reali. Il punto è riferito all'UCS corrente. La funzione è simile al **GETPOINT**. Un rettangolo elastico viene visualizzato sullo schermo durante lo spostamento del cursore; uno spigolo del rettangolo si trova sul *punto* indicato (fisso), mentre quello opposto si muove assieme al cursore. L'argomento *richiesta* è una stringa opzionale; se è presente viene visualizzata sotto forma di messaggio di richiesta per l'utente.
(**getdist** [punto] [richiesta])	introduce una pausa perché l'utente immetta una distanza (da tastiera o in modo dinamico), che poi viene restituita come numero reale. L'argomento *punto* è un punto opzionale, se è presente attiva l'inserimento della distanza in modo dinamico e viene interpretato come primo punto della linea elastica disegnata da AutoCAD sullo schermo. L'argomento *richiesta* è una stringa opzionale; se è presente viene visualizzata sotto forma di messaggio di richiesta per l'utente. L'utente può rispondere al messaggio di richiesta con un valore, con due punti oppure con il solo secondo punto se nella funzione è stato inserito l'argomento *punto*.
(**getenv** nome_variabil)	restituisce una stringa che rappresenta il valore assegnato ad una variabile di ambiente (come PATH, ACADCFG, ecc.).
(**getint** [richiesta])	introduce una pausa perché l'utente immetta un numero intero (da tastiera) e restituisce tale intero. L'argomento *richiesta* è una stringa opzionale; se è presente viene visualizzata sotto forma di messaggio di richiesta per l'utente.
(**getkword** [richiesta])	introduce una pausa perché l'utente possa immettere una parola chiave; restituisce la sequenza di caratteri immessa, sotto forma di stringa. Questa funzione, quando usata assieme a **INITGET** (vedi) permette di specificare quali risposte sono accettabili per un certo messaggio di richiesta rivolto all'utente.

(**getorient** [punto] [richiesta])	introduce una pausa perché l'utente immetta (da tastiera o in modo dinamico) un angolo, il quale poi viene restituito in radianti e relativo al piano XY corrente; vengono ignorati i valori delle variabili ANGDIR e ANGBASE (zero gradi sempre a destra e senso positivo antiorario). L'argomento *punto* è un punto 2D opzionale; se è presente attiva l'inserimento dell'angolo in modo dinamico e viene interpretato come primo punto della linea elastica disegnata da AutoCAD sullo schermo. L'argomento *richiesta* è una stringe opziona1e; se è presente viene visualizzata sotto forma di messaggio di richiesta per l'utente.
(**getpoint** [punto] [richiesta])	introduce una pausa perché l'utente immetta (da tastiera in modo dinamico) un punto, il quale poi viene restituito come lista di due o tre numeri reali. Il punto è riferito all'UCS corrente L'argomento *punto* è un punto 2D opzionale, se è presente attiva la visualizzazione di una linea elastica che partendo da tale punto raggiunge la posizione del cursore. L'argomento *richiesta* è una stringa opzionale; se è presente viene visualizzata sotto forma di messaggio di richiesta per l'utente.
(**getreal** [richiesta])	introduce una pausa perché l'utente immetta un numero reale e restituisce tale numero reale. L'argomento r*ichiesta* è una stringa opzionale; se è presente viene visualizzata sotto forma di messaggio di richiesta per l'utente.
(**getstring** [cr] [richiesta])	introduce una pausa perché l'utente immetta una stringa e restituisce tale stringa; se la stringa immessa supera i 132 caratteri solo i primi 132 saranno restituiti. Se la stringa contiene il carattere "\", questo viene automaticamente convertito in "\\". Se l'argomento *cr* e presente ed è diverso da **nil** la siringa immessa può contenere spazi e deve quindi essere terminata con ↵; in caso contrario il primo spazio immesso dall'utente viene interpretato come ↵, secondo lo standard AutoCAD.
(**getvar** variab_acad)	restituisce il valore attuale della variabile di sistema specificata, il cui nome deve essere scritto maiuscolo e fra virgolette. (getvar "CMDECHO") ⇒ 0 oppure 1
(graphscr)	passa dallo schermo testo a quello grafico (come tasto F2) e restituisce sempre **nil**

(initget [bit] [parole_ch])	questa funzione si adopera assieme ad una delle funzioni GETxxx già viste (escluse GETSTRING e GETVAR); essa, posta prima della funzione GETxxx cui si riferisce, permette di porre dei vincoli ai dati che la funzione GETxxx accetta da parte dell'utente. L'argomento *bit* è un numero intero che può assumere i seguenti valori: **1** non si accettano input nulli (non si può rispondere con il solo ↵) **2** non si accettano valori uguali a zero **4** non si accettano valori negativi **8** non si effettua il controllo dei limiti, anche se la variabile LIMCHECK è attiva **32** le eventuali linee elastiche generate dalla funzione GETxxx appaiono tratteggiate (non funziona se la variabile POPUPS è zero) **64** impedisce l'input di una coordinata Z per la funzione *getdist*; permette a un'applicazione di assicurare che tale funzione restituisca una distanza bidimensionale **128** permette un input qualunque e lo considera come una parola chiave, rispettando qualsiasi altro bit di controllo e parola chiave presenti nell'elenco. Questi valori possono anche essere combinati fra loro, ad esempio sostituendo al singolo numero una lista del tipo **(+ 1 2 4),** come in **(initget (+ 1 2 4))** L'argomento *parole_ch* (opzionale) è una stringa che definisce una sequenza di parole chiave separate da spazi; se essa é presente, nel caso in cui l'utente immetta non il dato atteso dalla funzione GETxxx (ad esempio un punto per **GETPOINT**), ma una stringa che corrisponde a una delle parole chiave indicate, tale parola viene restituita come risultato, sotto forma di stringa, dalla funzione GETxxx. Le parole chiave possono contenere delle lettere maiuscole; in tal caso sarà sufficiente che l'utente immetta le sole lettere maiuscole di una parola chiave affinché la funzione restituisca la parola intera. Il messaggio di richiesta della funzione GETxxx viene ripetuto fino a che i dati inseriti dall'utente non soddisfano le condizioni poste da *bit* o non corrispondono a qualcuna delle parole chiave eventualmente presenti. Tutte le impostazioni di **INITGET** valgono soltanto per la funzione GETxxx successiva e vengono poi abbandonate. La funzione restituisce sempre **nil**. …….. (initget 1 "Origine") (getpoint "Indicare il punto di partenza (Origine/<punto>): ") …….. In questo esempio se l'utente risponde al messaggio con una O maiuscola la finzione getpoint restituisce la stringa "*Origine*".

(**inters** pt1 pt2 p3 pt4 [sul_segmento])	restituisce il punto di intersezione dei due segmenti individuati dai quattro punti specificati (pt1 → pt2 = primo segmento, pt3 → pt4 = secondo segmento). Se i due segmenti non si intersecano, la funzione restituisce **nil.** L'argomento *sul_segmento* determina se il punto di intersezione debba essere cercato all'interno dei segmenti o sulle linee (di lunghezza infinita) che i segmenti individuano: *sul_segmento* assente ⇒ ricerca sui segmenti *sul_segmento* presente e /= **nil** ⇒ ricerca sui segmenti *sul_segmento* presente e = **nil** ⇒ ricerca sulle linee Tutti i punti sono valutati rispetto all'UCS corrente; se tutti i punti sono di tipo 3D, prima di cercare l'intersezione nel piano XY usando le loro proiezioni sullo stesso, il sistema cerca un'eventuale intersezione nello spazio.
(**osnap** pt stringa_modo)	restituisce un punto 3D risultante dall'applicazione dei modi di snap ad oggetto descritti dalla *stringa_modo* al punto *pt* indicato. Valgono le limitazioni indotte dalla vista 3D corrente. Il risultato può essere influenzato, come d'altronde accade quando si utilizza lo snap ad oggetto direttamente in AutoCAD, dalla vista 3D corrente e dalle dimensioni del mirino di snap ad oggetto (variabile APERTURE). **(osnap p1 "medio,fine")**
(**polar** pt angolo_distanza)	restituisce quel punto che presenta, rispetto al punto dato, le coordinate polari indicate da *angolo* e *distanza*. L'angolo va espresso in radianti, dal semiasse X positivo, con valori crescenti, in senso antiorario.
(**redraw**)	esegue un'operazione di ridisegno della finestra corrente.
(**setvar** nome_variabile valore)	assegna il valore indicato alla variabile di sistema AutoCAD specificata, il cui nome deve essere scritto maiuscolo e fra virgolette. Restituisce il valore assegnato come numero reale. (setvar "CMDECHO" 0) assegna alla variabile CMDECHO il valore zero e restituisce zero.
(**textscr**)	passa dallo schermo grafico a quello di testo (come tasto F2) e restituisce sempre **nil.**

Altre funzioni

(**alert** messaggio)	visualizza una casella di avviso che riporta il messaggio (stringa) indicato. Una casella di avviso é una finestra di dialogo contenente un unico pulsante OK. Nel caso si desideri che il messaggio sia ripartito su più righe bisogna includere nella stringa *messaggio* il carattere di riga nuova *\n* La lunghezza della riga ed il numero di righe di una casella di avviso dipendono dalla piattaforma dal dispositivo e dalla finestra utilizzata. AutoCAD tronca le stringhe che sono troppo lunghe per poter essere visualizzate in una casella di avviso.

(**atoms-family** formato [lista_simboli])	restituisce una lista di tutti i simboli (nomi di variabili e di funzioni) utilizzati nella sessione corrente, comprendendo sia i simboli predefiniti che quelli definiti dall'utente. Può essere utile per vedere se il nome che si pensa di assegnare ad una variabile è già in uso oppure è disponibile. L'argomento *formato* è un numero intero che può valere 0 oppure 1. Se vale 0 la funzione restituisce una lista di tutti i simboli in uso, mentre se il valore è 1 si ottiene una lista analoga in cui però ogni simbolo è convertito in una stringa L'argomento *lista_simboli* è una lista di stringhe che specificano i nomi dei simboli che si vogliono ricercare. Come esempio supponiamo di voler assegnare a due variabili i nomi *centro* e *distanza_centri*; si può controllare se questi nomi sono gia utilizzati scrivendo **(atoms-family O '("centro" "distanza_centri"))** Se la funzione restituisce *("centro" nil)* vuol dire che il nome *centro* è già utilizzato mentre *distanza_centri* è disponibile.
(**princ** [espressione])	visualizza sullo schermo l'espressione indicata. Restituisce la stringa visualizzata. **(princ "Fase di calcolo in esecuzione…")**
(**progn** espress…)	valuta, in modo sequenziale, ogni espressione indicata, e restituisce il valore dell'ultima espressione. Viene usata, per esempio, per fare in modo che la funzione **IF** esegua più operazioni.
(**prompt** messaggio)	visualizza sullo schermo il messaggio indicato (stringa di testo). Restituisce sempre **nil**. **(prompt "Fase di calcolo in esecuzione…")**
(**type** elemento)	mostra che tipo di dato è *elemento* **(type 12.45)** ⇒ REAL **(type "cipolla")** ⇒ STR
(**ver**)	restituisce una stringa che contiene il numero corrente della versione di AutoLISP

DEFINIZIONE DI FUNZIONI PERSONALIZZATE

Oltre ad impiegare le varie funzioni predefinite, in un programma AutoLISP solitamente si definiscono ed utilizzano delle ***funzioni personalizzate*** ognuna delle quali svolge un compito ben preciso. In pratica una funzione personalizzata non è altro che un insieme di espressioni LISP raggruppate sotto un certo nome di funzione; una volta definita la funzione si potrà ottenere l'esecuzione di tutte le istruzioni che essa contiene semplicemente richiamandola tramite il nome che le è stato assegnato.

Nell'ambito delle più o meno numerose funzioni personalizzate che si definiscono all'interno di un medesimo programma bisogna porre attenzione all'uso che si fa delle variabili. Dal momento che ogni variabile conserva sempre l'ultimo valore assegnatole, l'uso di variabili con lo stesso nome in funzioni diverse può essere opportuno in alcuni casi ed a volte invece fonte di risultati errati. A questo proposito è bene ricordare che al momento della definizione di una funzione personalizzata si possono definire delle ***variabili locali***; quando una variabile è stata dichiarata variabile locale per una certa funzione il suo valore viene automaticamente azzerato non appena la funzione è terminata. Le altre variabili (quelle non locali) rimangono invece condivise tra tutte le funzioni presenti in memoria.

(**defun** nome lista_argomenti espressioni…)	definisce una funzione e ne restituisce il nome. La *lista_argomenti* contiene i nomi degli eventuali argomenti della funzione ed anche delle eventuali variabili locali (separate dai primi da una barra obliqua). La lista deve essere sempre presente; nel caso in cui la funzione non abbia né argomenti né variabili locali la *lista_argormenti* è vuota e bisogna indicarla con due parentesi: "( )". Le *espressioni* che seguono la *lista_argomcnti* costituiscono la sostanza vera e propria della funzione che si definisce; esse verranno valutate in sequenza quando la funzione sarà richiamata. A titolo di esempio si considerino le espressioni seguenti: **(defun gradi_rad (angolo_in_gradi)** **(* pi (/ angolo_in_gradi 180.0))** **)** Questa funzione ha come scopo la conversione in radianti di un angolo espresso in gradi; l'argomento ***angolo_in_gradi*** viene diviso per 180 ed il risultato è moltiplicato per π. Una volta caricato in memoria (funzione **LOAD**) il file che contiene questa definizione di funzione, basta richiamare la funzione stessa, fra parentesi, in risposta al messaggio "*Comando:*", indicando anche il valore dell'argomento (in questo caso l'angolo da convertire): Comando: **(gr_rad 180)** la funzione viene valutata e restituisce il valore dell'ultima espressione che in questo caso è il numero reale 3.14159. ***DEFUN*** può essere usata anche per creare **nuovi comandi** richiamabili direttamente dal messaggio "*Comando:*"di AutoCAD, senza le parentesi. In questi casi il nome della funzione deve essere del tipo "***C:XXX***" in cui la parte "***C:***" è obbligatoria, mentre le lettere ***XXX*** vanno sostituite con un nome a piacere, purché non si tratti del nome di un'altra funzione o di un comando già definito. Tale nome viene poi usato per avviare il nuovo comando. Un comando definito mediante questo sistema può essere anche utilizzato in modalità trasparente purché nella funzione non venga usata la funzione ***COMMAND***. Di seguito sono riportati due esempi relativi alla creazione di una semplice funzione che richiama lo zoom dinamico; sono messi in evidenza i due diversi modi di lanciare la funzione a seconda del modo in cui essa è stata definita **(defun ZD () (command "zoom" "d"))** ⇒ Comando: **(ZD)** **(defun C:ZD () (command "zoom" "d"))** ⇒ Comando: **ZD**

CARICAMENTO DEI FILES DI PROGRAMMA

Le funzioni definite in un certo file AutoLISP non sono utilizzabili all'interno di AutoCAD se il file non è caricato in memoria. Tramite le funzioni descritte di seguito si possono di volta in volta caricare i programmi che interessano per poter poi lanciare le funzioni desiderate.

Esiste un particolare file, ***ACAD.LSP***, che viene caricato in memoria automaticamente ad ogni avvio di AutoCAD, oppure ogni volta che si entra in un disegno. In questo file risulta utile inserire le funzioni LISP che si vogliono avere sempre a disposizione durante le sessioni di lavoro. Dal momento che la ricerca ditale file viene fatta da AutoCAD seguendo i consueti percorsi di libreria a partire dalla directory del disegno, è possibile al limite definire diversi file ACAD.LSP, posizionati nelle varie directory di lavoro e diversi uno dall'altro in base alle differenti tipologie di disegno. Se nel file ACAD.LSP risulta definita, fra le altre, una funzione di nome **S::STARTUP** essa verrà eseguita automaticamente ogni volta che il file viene caricato, vale a dire ogni volta che si entra in un disegno.

(**load** nome_file [se_fallisce])	carica in memoria un file di espressioni AutoLISP e le valuta. L'argomento *nome_file* è una stringa che rappresenta il nome del file; se non viene indicata alcuna estensione ***LOAD*** ricerca l'estensione LSP. La stringa può includere l'indicazione del percorso relativo al file da caricare; nell'indicazione dei percorsi bisogna usare, per separare le directory, una singola barra diritta (/) oppure una coppia di barre rovesciate (\\). Se non si indica alcun percorso la ricerca del file viene fatta nei percorsi di libreria di AutoCAD. Nel caso in cui la funzione non abbia esito positivo essa restituisce il valore dell'argomento *se_fallisce*; se quest'ultimo non è presente un esito negativo di ***LOAD*** causa un messaggio di errore di AutoLISP. Quando il caricamento ha esito positivo la funzione restituisce il valore dell'ultima espressione presente nel file Ad ogni avviamento dell'editor grafico viene caricato automaticamente, se esiste, il file ACAD.LSP.

(**autoload** nome_file elenco_comandi)	abilita un meccanismo tale per cui il file LISP in cui è contenuta la definizione di un certo comando viene caricato automaticamente non appena il comando viene richiamato per la prima volta. L'argomento *nome_file* è una stringa che rappresenta il nome di un file LISP, cosi come accade per la funzione LOAD. L'argomento *elenco_comondi* è una lista i cui elementi sono i nomi, sotto forma di stringhe, dei comandi definiti nel file indicato in *nome_file.* Si supponga ad esempio che il file *FIGURE.LSP* contenga le definizioni delle funzioni *C:QUADRATO, C:ESAGONO, C:STELLA*, corrispondenti ad altrettanti comandi personalizzati. L'espressione **(autoload "figure" '("quadrato" "esagono" "stella"))** dice ad AutoLISP che, non appena sarà lanciato, da tastiera o da menu, uno dei comandi *QUADRATO, ESAGONO* e *STELLA*, bisognerà subito caricare il file *FIGURE.LSP* ed eseguire la funzione corrispondente ivi contenuta. Se nel file ACAD.LSP vengono incluse espressioni di questo tipo si avrà che i file che contengono i vari comandi personalizzati saranno caricati in memoria solamente quando i corrispondenti comandi verranno utilizzati. Questa funzione è disponibile solamente nella versione 13 e successive di AutoLISP.

INDICE

INTRODUZIONE AD AUTOLISP 1

LA PROGRAMMAZIONE 2

LINGUAGGI DI PROGRAMMAZIONE 4

SCRIVERE UN PROGRAMMA 6

Uso delle variabili 7

COLLAUDARE UN PROGRAMMA 9

IL LINGUAGGIO AUTOLISP 10

STRUTTURA DEL LINGUAGGIO 11

LA VALUTAZIONE DELLE ESPRESSIONI 13

STESURA DI UN PROGRAMMA 14

FUNZIONI PRINCIPALI DI AUTOLISP 16

La funzione di assegnazione 16

Funzioni matematiche 16

Funzioni di confronto e condizionali 19

Funzioni di manipolazione delle liste 21

Funzioni relative alle stringhe di testo 23

Funzioni specifiche per l'ambiente CAD 25

Altre funzioni 30

DEFINIZIONE DI FUNZIONI PERSONALIZZATE 32

CARICAMENTO DEI FILES DI PROGRAMMA 34

INDICE 36