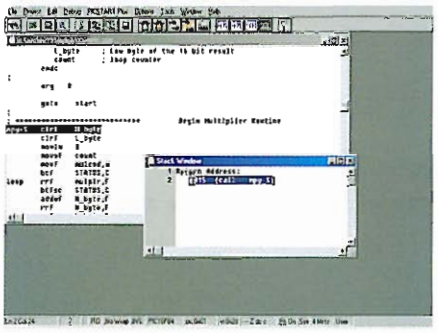
# Procedimenti e funzioni: annidamenti

All'interno del trattamento dei procedimenti e delle funzioni troviamo il concetto di annidamento, che per la sua importanza merita un capitolo a sé.

## Esecuzione di subroutine

Per fare in modo che una subroutine, oppure un procedimento o una funzione, vengano eseguiti per risolvere un compito è necessario eseguire una chiamata ad essi. In tutti i linguaggi di programmazione ci sono istruzioni di chiamata a subroutine, che alcune volte sono specifiche, mentre in altre bisogna utilizzare il nome della subroutine. In precedenza vi abbiamo già spiegato la sequenza degli avvenimenti che si sviluppano: chiamando una subroutine si salta a un'altra parte del programma, si eseguono le istruzioni che formano parte della subroutine, e si ritorna al punto di partenza (istruzione successiva a quella della chiamata) per continuare con il resto delle istruzioni o subroutine. Questo è possibile grazie al fatto che ogni istruzione ha assegnato un indirizzo di memoria di programma, e quando c'è una chiamata a una subroutine viene memorizzato l'indirizzo del punto in cui ci si trovava nel momento prima di saltare. Quando è necessario tornare si recupera quell'indirizzo.

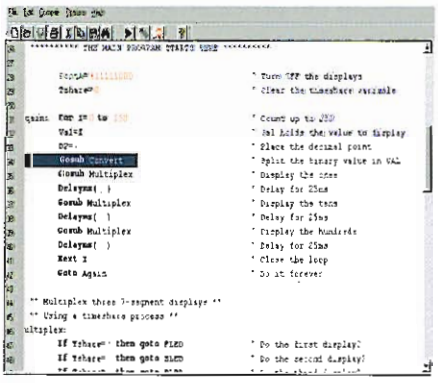
Gli indirizzi solitamente vengono scritti nello stack; lo stack è uno spazio di memoria il cui funzionamento lo possiamo comparare con una "pila di piatti": durante la scrittura li posizioniamo uno sopra l'altro, in modo che quando torneremo a recuperarli dovremo farlo nell'ordine inverso a quello di inserimento, utilizzando quello che è stato "impilato" per ultimo. Un programma nel suo stato puro potrebbe essere formato da una sequenza di subroutines e da un modulo principale (denominato "main" in alcuni linguaggi) da cui vengono eseguite solo le chiamate alle suddette subroutines.

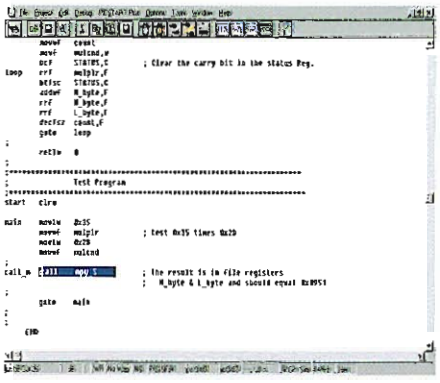


L'indirizzo di ritorno si scrive nello stack.

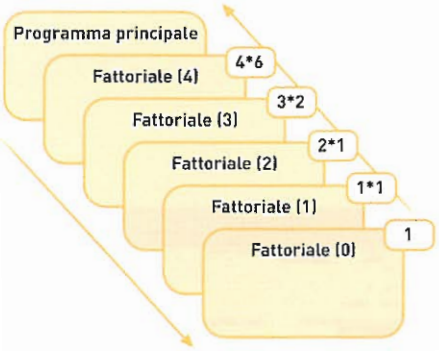
## Concetto di annidamento

Dall'interno di una subroutine si può eseguire una chiamata a un'altra subroutine. Se la chiamata è fatta alla stessa subroutine, stiamo parlando di una routine annidata. Il concetto non è semplice e bisogna essere molto scrupolosi nel suo utilizzo, inoltre, non tutto può essere risolto in questo modo. Infatti non è sufficiente che il linguaggio di programmazione permetta gli annidamenti, ma anche la routine stessa deve essere stata pensata in base a questo, dato che una semplice chiamata di una routine a se stessa non ci assicura che l'annidamento sia stato fatto in modo corretto. Per iniziare, così come in un ciclo, deve esistere una condizione di termine,





Esempio di chiamata a subroutine in due linguaggi di programmazione.




**Esempio di una routine annidata.**

oppure si genererà un ciclo infinito che fuoriuscirà dallo stack.

## Esempio di annidamento

In matematica esistono numerosi esempi di annidamenti. Uno dei primi esempi della sua applicazione è nella generazione di fattoriali. Per trovare il fattoriale di un numero bisogna moltiplicare questo numero per ognuno dei valori immediatamente precedenti, sino ad arrivare a zero, che non si moltiplica. In questo modo, il fattoriale del numero 4 sarà : 4!= 4*3*2*1*. Fare una routine in modo tradizionale per risolvere questo problema, è semplice, però in questo caso vogliamo pensare in forma annidata. Come possiamo notare, il fattoriale di un numero è lo stesso del fattoriale del numero precedente. Il termine si trova nel numero 0, dato che non richiede la chiamata successiva alla routine in quanto il fattoriale di 0 è 1.

Bisogna tuttavia tenere conto del numero di indirizzi di cui è composto lo stack, dato che questo indicherà la quantità di routines che si possono annidare o, in altre parole, il livello di recursività permesso. Devono essere inoltre tenuti in considerazione anche gli annidamenti indiretti, dato che con procedimenti che si richiamano l'uno con l'altro, vengono generati degli anelli, quindi anche in questo caso deve esistere una condizione di fermata. Un altro problema è che gli annidamenti hanno bisogno di molta memoria se utilizzati con grandi strutture di dati, e anche questo è un fattore da tenere sotto controllo.

## Passaggio di parametri per valore e per riferimento o traslazione

Quando abbiamo differenziato i procedimenti dalle funzioni, abbiamo detto che i primi non possono restituire dei valori. In realtà questo non è del tutto esatto, in quanto anche se non lo possono fare come una funzione, in forma "pulita", lo possono fare fornendo i risultati su variabili globali, utilizzabili da tutte le parti del programma, o in valori restituiti come riferimenti. Allo stesso modo delle variabili globali, i cambiamenti che si effettuano su di una variabile passata come riferimento all'interno di un procedimento, vengono mantenuti quando questo termina, quindi influenzeranno il programma principale e gli altri moduli che utilizzano questa variabile. Con i valori parametrizzati, tuttavia, all'interno del procedimento, viene utilizzata solamente una copia della variabile, quindi anche se ne viene modificato il valore, questo non influenzerà la variabile del programma principale.

Nell'esempio, la variabile "max" è fornita come riferimento, quindi "z" sarà modificata. Nel caso di un programma scritto nel modo tradizionale bisogna fare attenzione all'uso di variabili come riferimento, quando si programma in modo recursivo, cioè utilizzando gli annidamenti, il problema può risultare maggiore, in quanto non c'è controllo sulla parte del programma o iterazione che è in esecuzione in un determinato momento, né su un'eventuale modifica del valore di una variabile. In questi casi di solito si utilizzano funzioni al posto dei procedimenti.

**Esempio di variabile passata come riferimento.**

```
PROCEDURE maggiore (a, b: REAL; VAR max: REAL);
BEGIN
        IF a → b THEN max := a ELSE max:= b;
FINE;

BEGIN
REPEAT
        WRITELN ('Introdurre 2 numeri:  ');
        READLN (x, y);
        Maggiore (x, y, z);
        WRITELN ('Il numero maggiore è: ', z);
FINE.
```