Algoritmi e basi della programmazione

Fulvio Ferroni *fulvioferroni@teletu.it*

2010.08.31

Indice generale

Introduzione

Scopo di questa dispensa è quello di esaminare le nozioni fondamentali riguardanti la scrittura di programmi senza esaminare in dettaglio alcun linguaggio di programmazione.

Una scelta precisa viene invece effettuata a livello di modello di programmazione, a vantaggio del paradigma imperativo, senza che ciò impedisca un breve esame anche degli altri paradigmi.

Il linguaggio cui si fa riferimento, pur senza dettagliarne i costrutti e le istruzioni, è il c; per questo anche il linguaggio di progetto per la descrizione degli algoritmi, proposto nel prossimo capitolo e concepito autonomamente dall'autore, si ispira alla sintassi del linguaggio c.

La dispensa inizia con la teoria degli algoritmi, allargando il discorso all'attività di analisi da svolgere come primo passo del ciclo di vita dei programmi; si passa poi alle generalità sui paradigmi di programmazione, alla suddivisione in livelli dei linguaggi, alle modalità per la loro traduzione; successivamente si illustrano le più importanti strutture dati astratte e si conclude con un breve esame sulla documentazione e produzione del software.

Capitolo 2

Gli algoritmi

Il termine algoritmo è ispirato dal nome del matematico e astronomo arabo del VII° secolo Abu Ja Mohammed Ibn Musa Al-Khowarizmi che, oltre ad essere l'inventore dello zero e dell'algebra, ha la paternità di quello che è considerato uno dei primi algoritmi della storia: il metodo per sommare due numeri incolonnandoli e procedendo cifra per cifra da destra verso sinistra tenendo conto dei riporti.

Per '**algoritmo**' si può intendere un «insieme di istruzioni che definiscono una sequenza di operazioni mediante le quali si risolvono tutti i problemi di una certa classe»; in particolare è importante l'ultima parte della definizione dove si specifica che un algoritmo deve avere tra le sue caratteristiche quella della '**generalità**'.

Un'altra fra le numerose definizioni possibili può essere la seguente: «metodo di elaborazione da applicare a certi dati iniziali per ottenere dei dati finali o risultati».

In ogni caso è importante che, in partenza, il problema da risolvere sia «ben posto» e cioè che:

- sia chiaro l'obiettivo da raggiungere;
- i dati di partenza sia noti e sufficienti;
- il problema sia risolubile da parte di chi lo affronta.

Se manca una di queste condizioni l'algoritmo non può essere individuato.

Si deve anche notare come sia importante, quando si affronta un problema, non confondere il risultato (ciò che si vuole ottenere) con la soluzione (il metodo che conduce al risultato elaborando i dati di partenza).

L'individuazione dell'algoritmo risolutivo di un problema è solo il primo passo di un procedimento in più fasi che conduce alla soluzione del problema stesso in modo automatico con l'utilizzo del sistema di elaborazione.
Questo procedimento verrà dettagliato in seguito, ma possiamo già accennare che il passo successivo consiste nella trasformazione dell'algoritmo in un programma scritto usando un linguaggio di programmazione; si può affermare che l'algoritmo e il relativo programma sono due descrizioni, fatte con mezzi diversi, dello stesso metodo di soluzione.

2.1 Proprietà degli algoritmi

Distinguiamo da ora in poi l'aspetto della risoluzione del problema da quello dell'esecuzione del relativo algoritmo; distinguiamo cioè tra '**risolutore**' (l'uomo, che individua il metodo di soluzione del problema) ed '**esecutore**' (la macchina che deve eseguire tale metodo, descritto tramite un algoritmo).

L'algoritmo è solo la '**descrizione delle operazioni**' da svolgere; è quindi un'entità statica costituita da istruzioni, anch'esse statiche.

Quando l'algoritmo viene eseguito si ottiene un '**flusso d'esecuzione**' che invece è ovviamente dinamico ed è costituito da '**passi**' (ogni passo è l'esecuzione di un'istruzione).

A tale proposito può essere utile pensare alla differnza esistente tra una ricetta di una torta (descrizione delle operazioni da svolgere) e la sua «esecuzione» per la realizzazione della torta.

Nel contesto informatico l'esecutore, cioè il sistema di elaborazione, è una macchina non intelligente, capace solo di eseguire istruzioni molto elementari (anche se in modo rapidissimo), ma senza alcuna capacità critica o di ragionamento autonomo.

Un algoritmo deve allora avere le seguenti proprietà:

- '**generalità**', della quale abbiamo accennato in precedenza: non è un algoritmo il metodo che mi permette di stabilire se 19 è un numero primo, lo è invece il metodo che permette di decidere se un qualsiasi numero naturale è primo;
- '**comprensibilità**': l'algoritmo deve essere espresso in una forma comprensibile all'esecutore; questo non vuol dire che gli algoritmi debbano essere espressi nel linguaggio proprio della macchina, cioè quello binario, e neanche che si debbano subito utilizzare i linguaggi di programmazione, ma significa che si deve rispettare fin da subito un formalismo abbastanza rigido nella loro stesura;
- '**eseguibilità**': l'algoritmo deve anche essere eseguibile dalla macchina e quindi non deve prevedere operazioni ad essa sconosciute (l'insieme delle istruzioni eseguibili da un computer ha '**cardinalità finita**') o che non possano essere eseguite in un tempo finito (le istruzioni devono avere '**complessità finita**');
- '**finitezza**': l'algoritmo deve avere un numero finito di istruzioni;
- '**riproducibilità**': esecuzioni successive dell'algoritmo sugli stessi dati di input devono dare lo stesso risultato;
- '**non ambiguità**': non possono esserci istruzioni vaghe, che prevedano un comportamento probabilistico o la scelta casuale del prossimo passo da eseguire; la macchina non è infatti in grado di prendere decisioni complesse con ragionamento autonomo e quindi il metodo di soluzione deve essere '**completo**' (devono essere previste tutte le possibilità che possano verificarsi durante l'esecuzione) e '**deterministico**' (in caso contrario verrebbe meno anche la caratteristica della riproducibilità);
- '**discretezza**': l'esecuzione deve avvenire per passi discreti; i dati coinvolti nei calcoli assumono valori che cambiano tra un passo e il successivo e non assumono altri valori «intermedi»;
- '**non limitatezza dell'input**': non deve esserci un limite (almeno in linea teorica) alla lunghezza dei dati di input (e quindi anche alla capacità di memoria dell'esecutore);
- '**non limitatezza dei passi d'esecuzione**': devono essere possibili (almeno in linea teorica) esecuzioni costituite da un numero infinito di passi.

Le ultime due proprietà possono sembrare irrealistiche nelle applicazioni pratiche, ma sono necessarie per i seguenti motivi:

- se ci fosse un limite alla lunghezza dei dati di input verrebbe meno la proprietà della generalità: ad esempio non potremmo più concepire un algoritmo per decidere se un qualsiasi numero naturale è primo oppure no;
- esistono funzioni che si dimostra essere computabili a patto di ammettere un numero infinito di passi nell'algoritmo di calcolo.

Si deve comunque notare che queste grandi potenzialità teoriche degli algoritmi (input di lunghezza infinita e esecuzioni con infiniti passi) non sono sufficienti ad assicurare che tutte le funzioni siano computabili o, detto in altri termini, che tutti i problemi prevedano un algoritmo risolutivo.

Esiste ad esempio il famoso problema della «terminazione degli algoritmi»: si può dimostrare che «non esiste un algoritmo che permetta di stabilire se un qualsiasi altro algoritmo abbia termine».

Questi temi prettamente teorici inerenti la «teoria della computabilità» e che comprendono tra l'altro lo studio degli automi, della macchina di Turing, della «Tesi di Church-Turing«, pur molto interessanti, esulano dagli scopi di queste dispense e non vengono quindi ulteriormente trattati.

2.2 Rappresentazione degli algoritmi

Per quanto detto in precedenza appare del tutto impraticabile l'idea di affidare la descrizione di un algoritmo all'uso del linguaggio naturale (nel nostro caso la lingua italiana) con il rischio di introdurre ambiguità dovute alla presenza di sinonimi, omonimie, modi diversi di intendere la stessa frase.

Si pensi ad esempio alle possibili diverse interpretazioni che si possono dare alle seguenti frasi: «succede al Sabato», «lavoro solo da due anni», «mi piace molto la pesca».

Le interpretazioni e le sfumature delle frasi espresse in linguaggio naturale, spesso dipendenti dal contesto, possono essere colte solo dalla mente umana che è un «esecutore» molto più sofisticato, versatile ed elastico di qualsiasi macchina.

Per descrivere gli algoritmi si deve quindi ricorrere a '**linguaggi formali**' creati appositamente (e quindi artificiali); in queste dispense prenderemo in considerazione due esempi:

- un linguaggio '**lineare**' basato sul testo, spesso denominato '**linguaggio di progetto**';
- un linguaggio '**grafico**' basato su simboli chiamato '**diagramma di flusso**' (*flow chart*) o '**diagramma a blocchi**'.

Notiamo che, qualsiasi sia il metodo utilizzato per la rappresentazione dell'algoritmo, vale sempre la regola che le operazioni da esso descritte vengono eseguite una alla volta nell'ordine in cui sono scritte (a meno che non intervengano costrutti di programmazione particolari che alterano il normale flusso di esecuzione).

Prima di illustrare i due metodi di rappresentazione degli algoritmi è importante chiarire natura e ruolo delle variabili.

2.2.1 Concetto di variabile

Per '**variabile**' si intende un oggetto che ha un nome e che permette di immagazzinare e conservare un determinato valore; l'insieme delle variabili utilizzate in un certo algoritmo prende il nome di '**area di lavoro**' relativa ad esso.

In generale una variabile è caratterizzata da tre elementi:

- '**`tipo`**': indica se la variabile è un valore intero, reale, un carattere e così via;
- '**`nome`**': deve essere univoco e «significativo» (cioè inerente al ruolo che la variabile ricopre nell'elaborazione);
- '**`contenuto`**': il valore che in un certo passo dell'elaborazione è assegnato alla variabile.

In un algoritmo possono essere poi presenti altri oggetti, detti '**`costanti`**', che sono invece delle entità non modificabili.

consideriamo ad esempio l'algoritmo (per il momento descritto con normali frasi della lingua italiana) per calcolare il perimetro di un quadrato di lato 5:

- esegui l'operazione 5*4;
- poni il risultato nella variabile P;
- visualizza (o stampa) il valore del risultato P.

In questo esempio vengono usate la variabile ***P*** e le costanti ***4*** e ***5***.

Si noti in particolare la natura della seconda istruzione («poni ...») che è chiamata '**`istruzione di assegnazione`**' ed è «distruttiva» in quanto, assegnando un nuovo valore a ***P*** cancella il contenuto precedente di tale variabile.

L'algoritmo appena descritto non è degno neppure di essere considerato tale perché manca del tutto di generalità; una versione migliore è la seguente, ottenuta sostituendo una variabile ad una costante in modo da ottenere il metodo di calcolo del perimetro di un quadrato di lato ***L*** qualsiasi:

- acquisisci in input (leggi) il valore del lato ponendolo in L;
- esegui l'operazione L*4;
- poni il risultato nella variabile P;
- stampa il valore di P.

In questo caso abbiamo la presenza delle variabili ***P*** ed ***L*** e della costante ***4***.

Sostituendo anche a quest'ultima con una una variabile si ha una versione ancora più generale dell'algoritmo, che permette di calcolare il perimetro di un poligono regolare di ***N*** lati lunghi ***L***:

- leggi il numero dei lati e la misura del lato ponendoli in N e L;
- esegui l'operazione L*N;
- poni il risultato nella variabile P;
- stampa il valore di P.

2.2.2 Linguaggio di progetto

Il linguaggio di progetto che utilizziamo non fa parte di quelli ufficialmente riconosciuti come lo «Pseudo Pascal» o i linguaggi «L2P» e «L3P» creati dal professor Callegarin, ma è definito a scopo dimostrativo solo per queste dispense.

Si tratta di linguaggio basato su alcune parole della lingua italiana, sul simbolo di assegnazione «=», sui normali operatori aritmetici e di confronto.

Per quanto riguarda la sintassi delle istruzioni si fa riferimento a quella del linguaggio di programmazione c prevedendo quindi:

- l'uso del «;» per terminare ogni istruzione;
- i simboli «{» e «}» per iniziare e terminare i blocchi di istruzioni sequenziali;
- l'uso degli operatori «==», «!=» rispettivamente per l'uguaglianza e la disuguaglianza;
- l'uso degli operatori logici «!», «&&», «||» corrispondenti al NOT, AND, OR, rispettivamente;
- l'uso delle parentesi tonde per racchiudere i controlli logici.

Con questo strumento dobbiamo descrivere tutte le normali fasi di un algoritmo che sono:

- '**`inizio`**' dell'algoritmo: con la parentesi graffa aperta;
- '**`dichiarazione delle variabili`**': effettuata usando le parole ***intero***, ***reale***, ***carattere***, ***stringa*** seguite dal nome della variabile o delle variabili separati da virgole;
- '**`acquisizione dei dati di input`**': con l'istruzione ***leggi*** e le variabili di input indicate tra parentesi;
- '**`elaborazioni varie`**': con gli opertaori aritmetici, logici e il simbolo di assegnazione «=»;
- '**`emissione dei risultati`**': istruzione ***stampa*** e le variabili di input indicate tra parentesi;
- '**`fine`**' dell'algoritmo: con parentesi graffa chiusa.

Come esempio riscriviamo l'algoritmo per il perimetro del poligono regolare:

```
{
intero L,N,P;
leggi (N,L);
P = N*L;
stampa(P);
}
```

Torniamo ancora brevemente sull'istruzione di assegnazione P = N*L; la logica di un'istruzione di questo tipo è la seguente:

- a sinistra del simbolo di assegnazione ci può essere solo una variabile (non una costante o un'espressione di qualche genere);

- a destra del simbolo di assegnazione può trovarsi una variabile, una costante o un'espressione di qualsiasi genere;
- viene valutato l'oggetto presente a sinistra e il valore risultante viene assegnato alla variabile a destra, sovrascrivendo il valore precedentemente contenuto in essa.

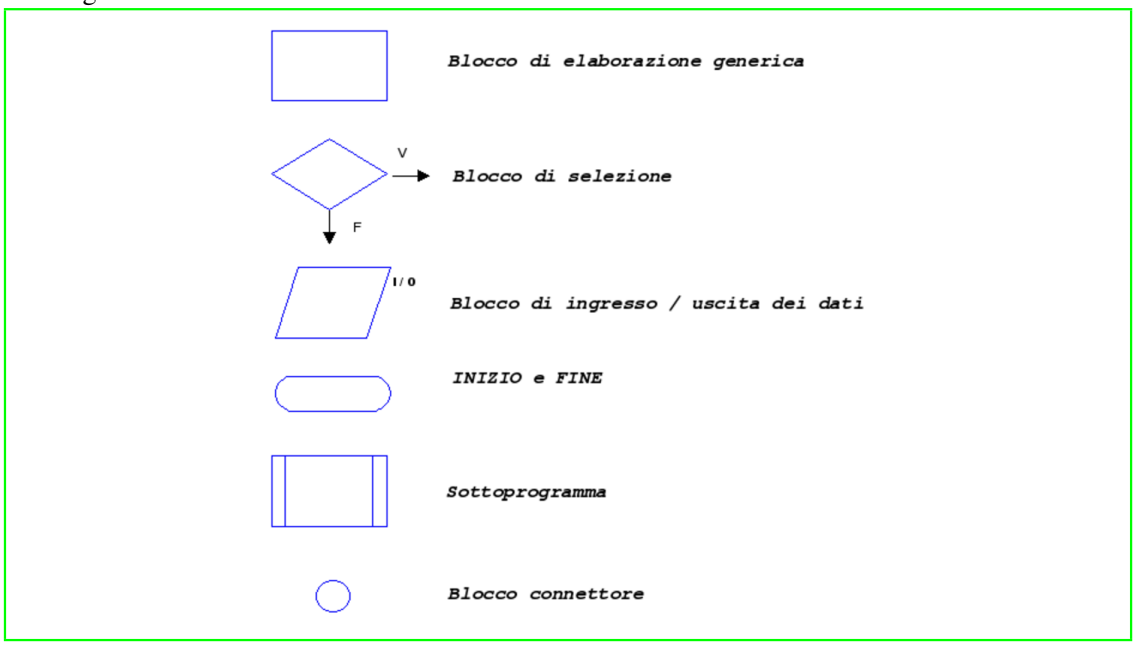
2.2.3 Diagrammi di flusso

I '**`diagrammi di flusso`**' permettono di descrivere gli algoritmi basandosi su simboli grafici contenenti le operazioni da eseguire.

I simboli si chiamano appunto blocchi e si differenziano per la loro forma in base alla funzione che svolgono nella descrizione dell'algoritmo; sono uniti da archi orientati (frecce) che permettono di esprimere la direzione del flusso di elaborazione.

Nella figura 2.2 è riportata la lista dei blocchi fondamentali con indicazione del ruolo svolto all'interno del diagramma.

Figura 2.2

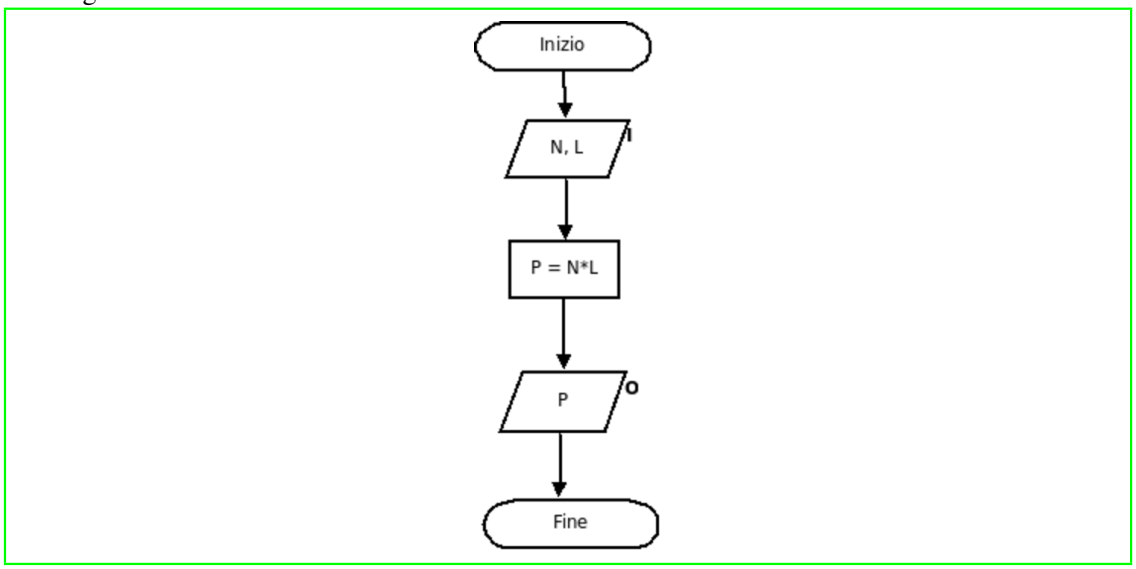


Dal blocco di inizio avremo sempre una freccia uscente, nel blocco di fine solo una freccia entrante; negli altri blocchi una freccia entrante ed una uscente ad eccezione del blocco di decisione che in uscita prevede due flussi.

Un caso a parte è il blocco connettore che non ha un ruolo effettivo nella descrizione dell'algoritmo ma serve solo ad unire parti diverse di un diagramma suddiviso per motivi di spazio.

Come esempio di primo banale diagramma di flusso, vediamo ancora una volta l'algoritmo per il calcolo del perimetro del poligono regolare, mostrato nella figura 2.3.

Figura 2.3



2.3 Strutture di controllo fondamentali degli algoritmi

Le istruzioni che compongono un algoritmo sono organizzate in '**strutture**' che permettono di avere un '**controllo**' sul flusso di elaborazione.

Le '**strutture di controlo fondamentali**', cioè quelle grazie alle quali si può descrivere qualsiasi algoritmo, sono:

- '**sequenza**';
- '**selezione**';
- '**iterazione**' o '**ciclo**'.

2.3.1 Sequenza

La sequenza è una struttura molto banale e presente in qualsiasi algoritmo in quanto contiene semplicemente un successione di istruzioni che devono essere eseguite nell'ordine in cui sono elencate e questo corrisponde appunto alla natura di un qualsiasi algoritmo.

Negli algoritmi meno banali avremo però la presenza di strutture di sequenza all'interno di altri tipi di strutture.

L'inizio e la fine di una sequenza si indicano nel linguaggio di progetto con la parentesi graffa aperta e chiusa mentre nei diagrammi di flusso non sono previsti simboli particolari.

L'algoritmo del perimetro del poligono regolare è un esempio in cui è presente solo la struttura di sequenza; come ulteriore esempio vediamo, con il linguaggio di progetto, l'algoritmo per lo scambio del valore di due variabili:

```
{
intero A,B,C;
leggi (A, B);
C = A;
A = B;
B = C;
stampa(A, B);
}
```

In questo algoritmo la variabile ***C*** non è né di input né di output ma è comunque indispensabile per l'elaborazione che deve essere svolta; si dice allora che è una '**variabile di appoggio**'.

2.3.2 Selezione

La struttura, o '**costrutto**' di selezione permette di:

- eseguire una o più istruzioni al verificarsi di una certa condizione; in tal caso si parla di '**selezione a una via**';
- eseguire una o più istruzioni al verificarsi di una certa condizione ed altre istruzioni se non si verifica; in questo caso si parla di '**selezione a due vie**'.

Nel linguaggio di progetto il costrutto di selezione a una via si indica con:

```
se (condizione) {

    sequenza di istruzioni

}
```

Mentre quello di selezione a due vie con:

```
se (condizione) {

    sequenza1 di istruzioni

}
altrimenti {

    sequenza2 di istruzioni

}
```

Osservando questi primi costrutti si può già notare come le strutture di controllo possano essere '**annidate**' le une nelle altre: infatti all'interno dei rami delle istruzioni di selezione si trovano delle «sequenze» di istruzioni (che possono comunque essere costituite anche da una sola istruzione).

Come esempio consideriamo il problema dell'individuazione della soluzione di un'equazione di primo grado, a*x = b, scrivendo il relativo algortimo in linguaggio di progetto:

```
{
   intero a,b,x;
   leggi (a,b);
   se ( (a==0) && (b==0) ) {
      stampa ("Equazione indeterminata");
   }
   se ( (a==0) && (b!=0) ) {
      stampa ("Equazione impossibile");
   }
   se (a!=0) {
```

```
        x = b/a;
        stampa (x);
    }
}
```

In questo esempio si è fatto uso di un'importante convenzione per la scrittura dell'algoritmo che è l'‘**indentazione**’; si tratta di scrivere leggermente spostate verso destra (di qualche spazio o di una tabulazione) le sequenze di istruzioni annidate dentro altre strutture allo scopo di migliorare la leggibilità dell'algoritmo.

Si tratta di una tecnica molto utile, anche e soprattutto quando gli algoritmi diventano programmi e si devono scrivere sorgenti, spesso molto lunghi e complessi, usando un qualche linguaggio di programmazione.

Nell'esempio è stata indentata anche la sequenza principale dell'algoritmo, quella contenuta tra le parentesi graffe di inizio e fine.

Naturalmente l'indentazione non è obbligatoria né nel linguaggio di progetto né usando i linguaggi di programmazione veri e propri, ma è fortemente consigliata.

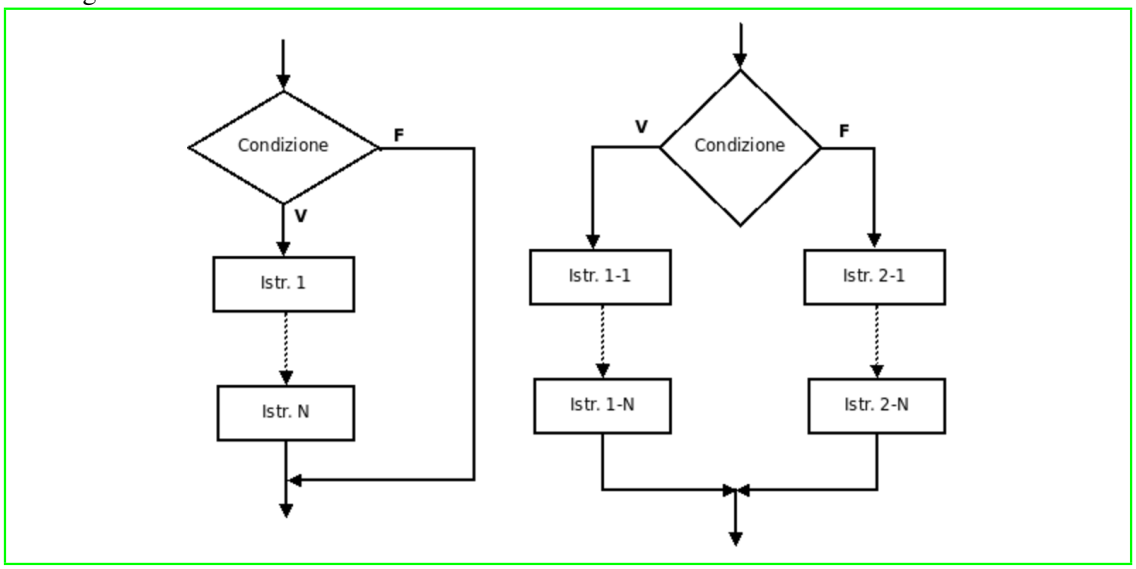
I suoi effetti benefici si apprezzano meglio riscrivendo l'algoritmo precedente usando strutture di selezione a due vie annidate invece che una sequenza di tre strutture di selezione a una via:

```
{
   intero a,b,x;
   leggi (a,b);
   se ( (a==0) && (b==0) ) {
      stampa ("Equazione indeterminata");
   }
   altrimenti {
      se ( (a==0) && (b!=0) ) {
         stampa ("Equazione impossibile");
      }
      altrimenti {
         x = b/a;
         stampa (x);
      }
   }
}
```

Il fatto che si siano scritti due algoritmi per la soluzione di uno stesso problema è da considerare del tutto normale (e anzi, ce ne sono anche diversi altri); infatti i metodi di soluzione possono essere vari e differenti l'uno dall'altro, sarà compito del bravo programmatore individuare la soluzione migliore tra quelle possibili.

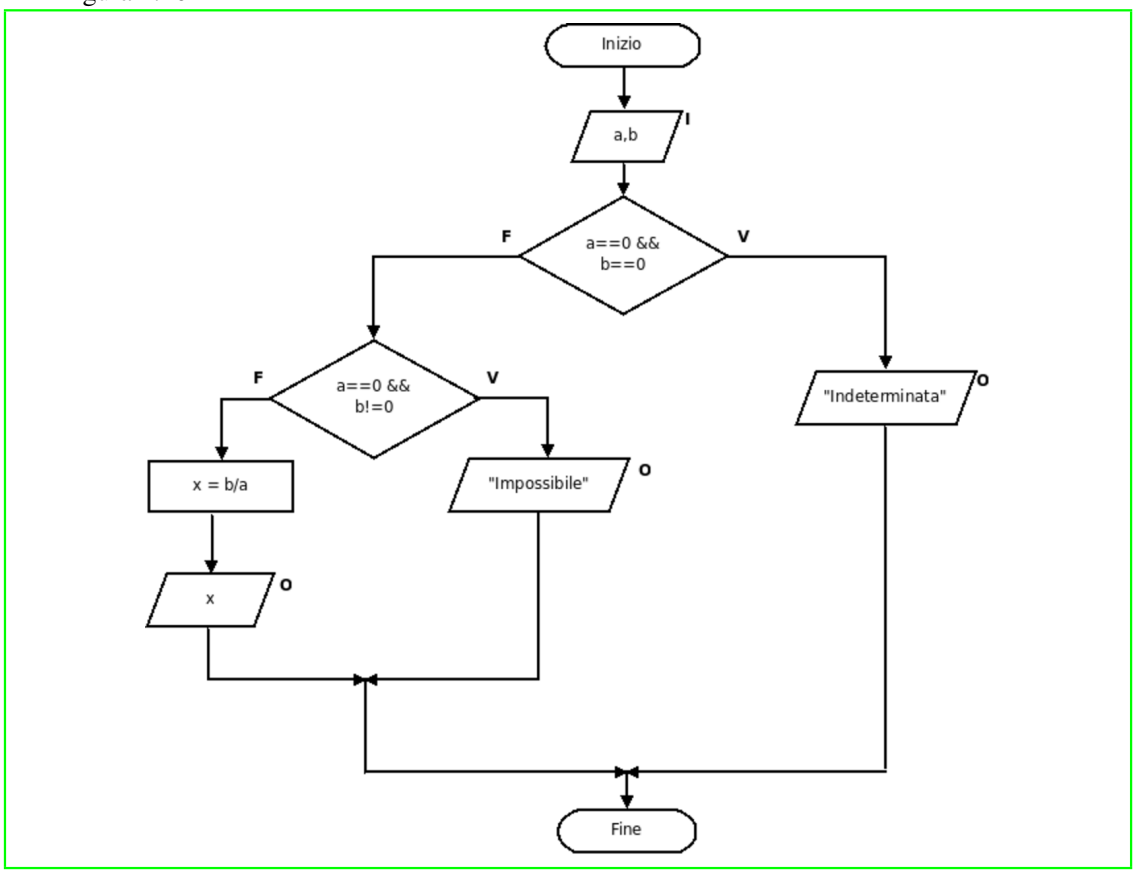
Per quanto riguarda i diagrammi di flusso, non esiste un blocco particolare per designare il costrutto di selezione; si fa uso, in modo opportuno, del blocco di decisione, come mostrato nella figura 2.9 (a sinistra abbiamo la selezione a una via, a destra quella a due vie).

Figura 2.9



Nella successiva figura 2.10 vediamo l'algoritmo per la soluzione dell'equazione di primo grado nella «seconda versione» (quella con le selezioni annidate).

Figura 2.10



Concludiamo questa breve introduzione sul costrutto di selezione osservando come il suo utilizzo permetta di scrivere algoritmi «completi» secondo la definizione data nel paragrafo 2.1.

In realtà sarebbe meglio dire che tale costrutto permette di «tentare» di scrivere algoritmi che tengano conto di tutte le eventualità possibili al momento dell'esecuzione, ad esempio al variare dei dati in input; questa infatti non è un'operazione semplice, specie in presenza di problemi di una certa consistenza, ed è uno dei principali motivi di malfunzionamento di molti programmi.

2.3.3 Iterazione

La struttura, o costrutto, di iterazione (o ciclo) permette di ripetere una o più istruzioni in sequenza per un numero potenzialmente infinito di volte.

Grazie alla sua presenza è possibile che gli algoritmi pur avendo la proprieta della «finitezza» possano (in linea teorica) prevedere esecuzioni con un numero infinito di passi (come detto nel paragrafo 2.1).

La ripetizione del gruppo di istruzioni soggette alla iterazione è soggetta al verificarsi di una certa condizione ed il controllo su di essa può essere in «testa» alla struttura oppure in «coda» alla stessa.

Ci sono quindi due tipi di iterazione:

- '**iterazione con controllo in testa**';
- '**iterazione con controllo in coda**'.

Nel linguaggio di progetto i due tipi vengono realizzati nel modo seguente:

```
mentre (condizione) {

    sequenza di istruzioni

}
```

```
esegui {

    sequenza di istruzioni

}
mentre (condizione);
```

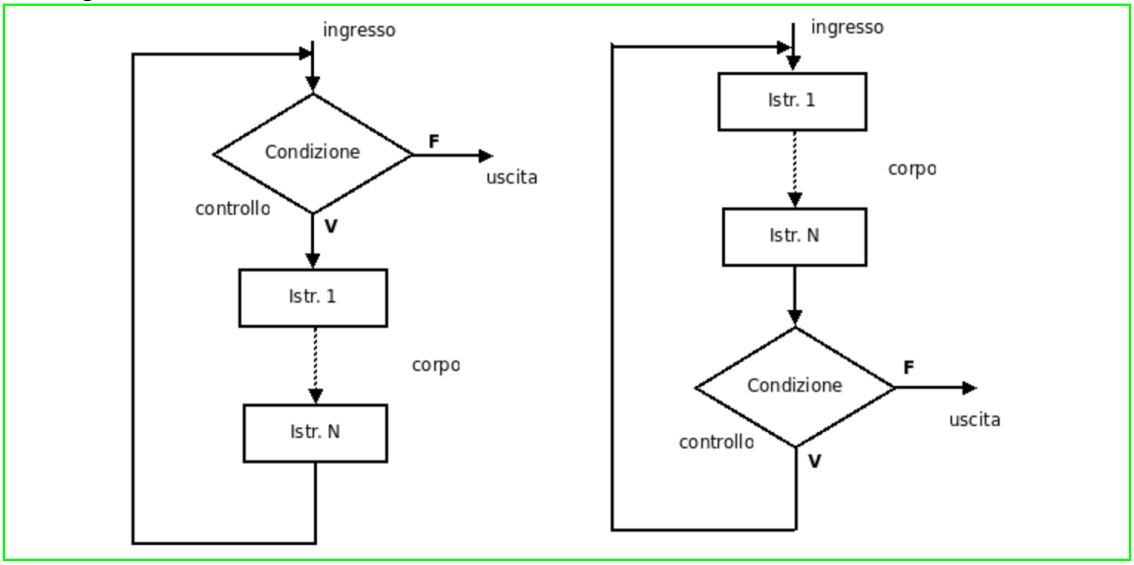
In entrambi i casi siamo in presenza di un '**corpo del ciclo**', che corrisponde alla sequenza di istruzioni (anche una sola) da ripetere, e di un '**controllo del ciclo**' rappresentato dalla condizione posta in testa o in coda al costrutto.

Il '**criterio di arresto**' del ciclo (o l'**uscita**' dal ciclo) si ha quando la condizione cessa di essere vera; si dice quindi che entrambi questi tipi di iterazione sono '**per vero**'.
La fondamentale differenza tra i due costrutti di ciclo è nel fatto che il corpo del ciclo con controllo in coda viene sempre eseguito almeno una volta.

Come per la selezione, anche per l'iterazione non esistono blocchi particolari da usare nei diagrammi di flusso; si deve ancora una volta usare in modo appropriato il blocco di decisione.

Nella figura 2.13 vediamo a sinistra il ciclo con decisione in testa e a destra quello con decisione in coda; vengono indicati i componenti dei costrutti, controllo e corpo, e i punti di ingresso e uscita dai costrutti stessi.

Figura 2.13



Prima di vedere esempi riguardanti i cicli introduciamo l'uso di alcuni tipi di variabili che hanno un ruolo importante in molti algoritmi iterativi:

- si dice '**`contatore`**' una variabile che si usa per contare il numero di volte che si è eseguita una certa iterazione; si inizializza di solito a 0 oppure a 1, secondo come si imposta il ciclo;
- si dice '**`accumulatore`**' una variabile che si usa per contenere il risultato di operazioni (somme o prodotti) successive; si imposta a zero se si devono accumulare somme, a 1 se si accumulano prodotti.

Come primo esempio vediamo l'algoritmo per il calcolo del prodotto tra due interi positivi ***a*** e ***b*** come successione di somme:

```
1   {
2      intero a,b,p,cont;
3      leggi (a,b);
4      p = 0;
5      cont = 0;
6      mentre (cont < b) {
7         p = p + a;
8         cont = cont + 1;
9      }
10     stampa(p);
11  }
```

In questo listato sono stati aggiunti i numeri di riga in modo da facilitare la stesura della '**`tavola di traccia`**' che è uno strumento grazie al quale si può esaminare il flusso di elaborazione dell'algoritmo e verificarne la correttezza.

Si tratta di una tabella in cui si indicano nelle colonne i nomi delle variabili (almeno le più importanti e significative per l'elaborazione) e nelle righe i passi dell'algoritmo che via via si svolgono partendo da valori di input scelti a piacere; può essere utile inserire anche una colonna in cui si indica il valore di verità della condizione da cui dipende l'esecuzione dell'iterazione.

Quando si esegue la tavola di traccia si deve avere l'accortezza di esaminare il comportamento dell'algoritmo anche quando i valori di ingresso sono '**`critici`**' o '**`ai limiti`**'; nel caso del

prodotto avremo tale situazione per ***a*** uguale a zero, ***b*** uguale a zero o entrambi uguali a zero.

Vediamo una prima tavola di traccia del nostro algoritmo scegliendo due valori «normali» delle variabili, ad esempio ***a=5***, ***b=3***:

Passo	Istr.	cont	p	contr. ciclo
1	3	-	-	-
2	4	-	0	-
3	5	0	0	-
4	6	0	0	V
5	7	0	5	V
6	8	1	5	V
7	6	1	5	V
8	7	1	10	V
9	8	2	10	V
10	6	2	10	V
11	7	2	15	V
12	8	3	15	V
13	6	3	15	F
14	10	3	15	F

Il valore di output è, correttamente, 15.

Adesso compiliamo la tavola di traccia di uno dei casi critici, ad esempio con ***a=7*** e ***b=0***:

Passo	Istr.	cont	p	contr. ciclo
1	3	-	-	-
2	4	-	0	-
3	5	0	0	-
4	6	0	0	F
5	10	0	0	F

Il valore di output è, correttamente, 0.

Infine vediamo un altro caso abbastanza particolare con ***a=8*** e ***b=1***:

Passo	Istr.	cont	p	contr. ciclo
1	3	-	-	-
2	4	-	0	-
3	5	0	0	-
4	6	0	0	V
5	7	0	8	V
6	8	1	8	V
7	6	1	8	F
8	10	1	8	F

Il valore di output è, correttamente, 8.

Come ulteriore esempio vediamo un algoritmo simile per il calcolo dell'elevamento a potenza come successione di prodotti.

In questo caso introduciamo una prima forma rudimentale di '**`controllo dell'input`**'; vogliamo cioè verificare che i valori di partenza siano: la base ***a*** positiva e l'esponente ***b*** non negativo (il fatto che siano entrambi interi viene invece ancora dato per scontato e non viene controllato).

```
1   {
2      intero a,b,p,cont;
3      esegui {
4         leggi (a,b);
5      }
6      mentre ( (a <= 0) || (b < 0) );
7      p = 1;
8      cont = 0;
9      mentre (cont < b) {
10        p = p * a;
11        cont = cont + 1;
12     }
13     stampa(p);
14  }
```

La verifica viene fatta con l'uso di un ciclo con controllo in coda che fa ripetere l'input finché i valori non sono corretti; si noti come sia indispensabile l'uso di questo tipo di ciclo in quanto almeno una volta le istruzioni del corpo devono essere eseguite.

Altra osservazione importante riguarda il valore di partenza dell'accumulatore ***p*** che è 1 in quanto l'accumulo avviene tramite dei prodotti.

Anche in questo caso eseguiamo la tavola di traccia con i valori ***a=5***, ***b=3*** prendendo in esame solo i passi successivi al controllo dell'input che non influiscono sul calcolo della potenza:

Passo	Istr.	cont	p	contr. ciclo
1	7	-	1	-
2	8	0	1	-
3	9	0	1	V
4	10	0	5	-
5	11	1	5	-
6	9	1	5	V
7	10	1	25	V
8	11	2	25	V
9	9	2	25	V
10	10	2	125	V
11	11	3	125	V
12	9	3	125	F
13	13	3	125	F

Il valore di output è, correttamente, 125.

Adesso compiliamo la tavola di traccia del caso critico più significativo, quello con ***b=0*** (e ***a*** qualsiasi, ad esempio, 12):

Passo	Istr.	cont	p	contr. ciclo
1	7	-	1	-
2	8	0	1	-
3	9	0	1	F
4	13	0	1	F

Il valore di output è, correttamente, 1.

Il successivo esempio riguarda l'individuazione del massimo tra un certo numero di valori immessi in successione grazie un apposito ciclo.

```
{
   intero val, max;
   carattere risposta;
   leggi (val);
   max = val;
   esegui {
      leggi (val);
      se (val > max) {
         max = val;
      }
      stampa ("Vuoi continuare (s/n)");
      leggi (risposta);
   }
   mentre ( (risposta != 'n') && (risposta != 'N') );
   stampa(max);
}
```

In questo esempio si può evidenziare l'uso della tecnica di '**elaborazione fuori ciclo**' che permette di inserire un primo valore che viene impostato come massimo; successivamente, tramite un ciclo con controllo in coda, si inseriscono gli altri valori uno alla volta facendo per ognuno il confronto con il valore massimo e adeguando quest'ultimo se necessario.

Come ultimo esempio vediamo l'algoritmo per il calcolo della media di ***n*** valori.

```
1   {
2      intero n, val, cont, somma;
3      reale media;
4      esegui {
5         stampa ("Numero di valori desiderati ");
6         leggi (n);
7      }
8      mentre (n <= 0);
9      somma = 0;
10     cont = 0;
11     mentre (cont < n) {
12        stampa("Inserire un valore ");
13        leggi (val);
14        somma = somma + val;
15        cont = cont + 1;
16     }
17     media = somma/n;
18     stampa (media);
19  }
```

In questo algoritmo si fa uso di un ciclo con controllo in coda per inserire, in modo controllato, la quantità di valori su cui impostare il ciclo principale (righe da 4 a 8); il ciclo non si conclude se il valore inserito non è maggiore di zero e questo permette, appunto, di avere un piccolo controllo sul valore ***n***.

Successivamente si esegue il ciclo, ad ogni passo del quale il valore immesso viene accumulato nella variabile somma (righe da 11 a 16).

Infine, esauritosi il ciclo, si calcola e si stampa a video la media (righe 17 e 18).

In questo caso vediamo anche la tavola di traccia considerando solo la parte più significativa dell'algoritmo, cioè quella che comprende il ciclo principale di elaborazione; i valori della variabile ***val*** sono indicati in tabella al momento dell'esecuzione della relativa istruzione di input, mentre per ***n*** supponiamo venga immesso il valore ***3***:

Passo	Istr.	somma	media	val	cont	contr. ciclo
1	9	0	-	-	-	-
2	10	0	-	-	0	-
3	11	0	-	-	0	V
4	12-13	0	-	7	0	V
5	14	7	-	7	0	V
6	15	7	-	7	1	V
7	11	7	-	7	1	V
8	12-13	7	-	3	1	V
9	14	10	-	3	1	V
10	15	10	-	3	2	V
11	11	10	-	3	2	V
12	12-13	10	-	8	2	V
13	14	18	-	8	2	V
14	15	18	-	8	3	V
16	11	18	-	8	3	F
17	17	18	6	8	3	F

Concludiamo il paragrafo con i diagrammi di flusso relativi all'algoritmo della potenza calcolata come successione di prodotti e al calcolo della media di ***n*** valori, illustrati rispettivanmente nelle figure 2.24 e 2.25.

Figura 2.24: potenza come succ. di prodotti

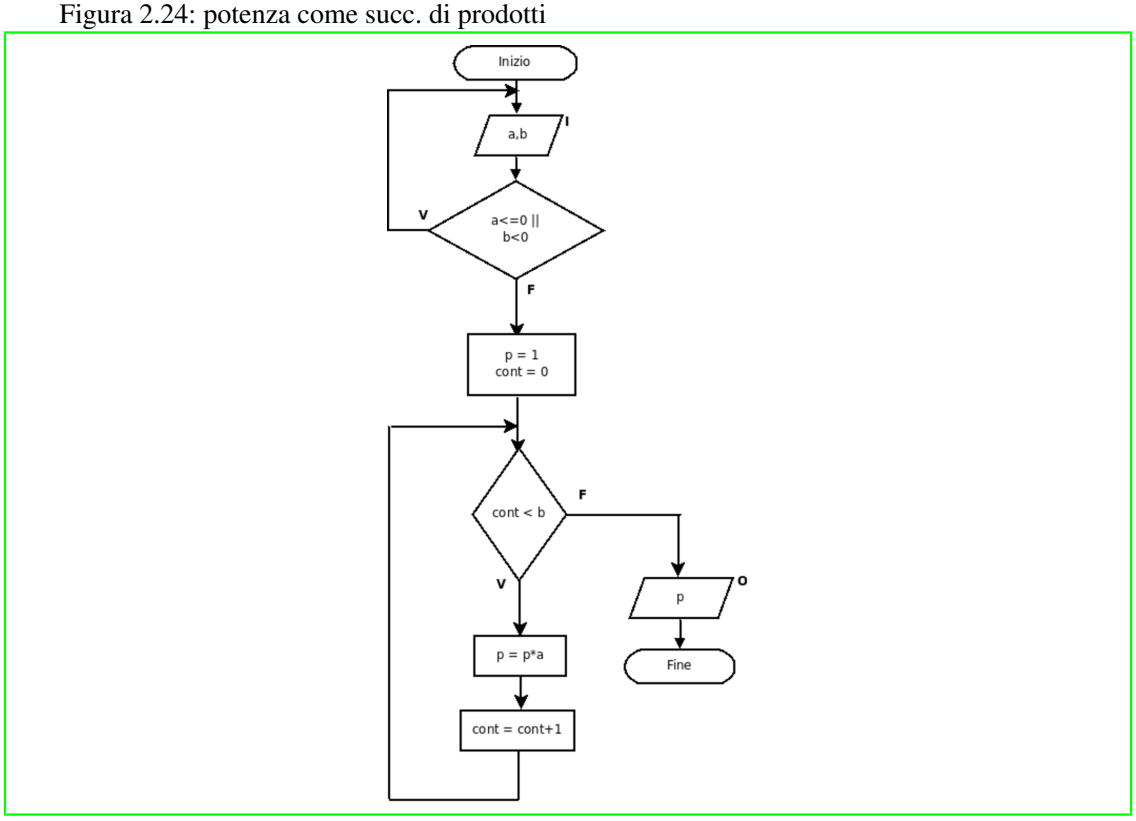
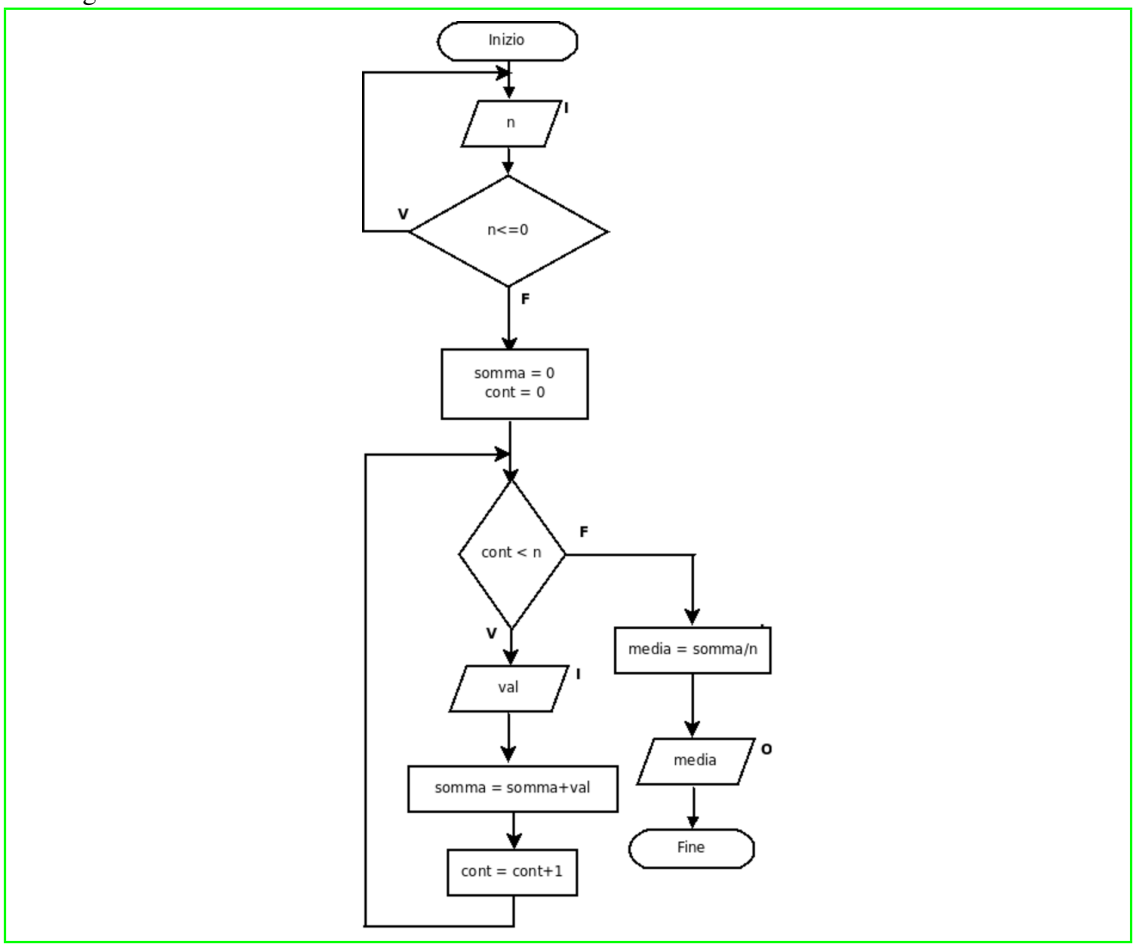


Figura 2.25: media di n valori



2.4 Strutture di controllo derivate degli algoritmi

Le '**`strutture di controllo derivate`**' hanno questo nome in quanto si possono ottenere dal quelle fondamentali, e ne estendono le potenzialità, ma non sono essenziali per la stesura degli algoritmi.

Esistono due strutture derivate:

- '**`selezione multipla`**';
- '**`iterazione calcolata`**'.

2.4.1 Selezione multipla

Con la selezione multipla si possono eseguire sequenze diverse di istruzioni in base al valore di una certa variabile intera.

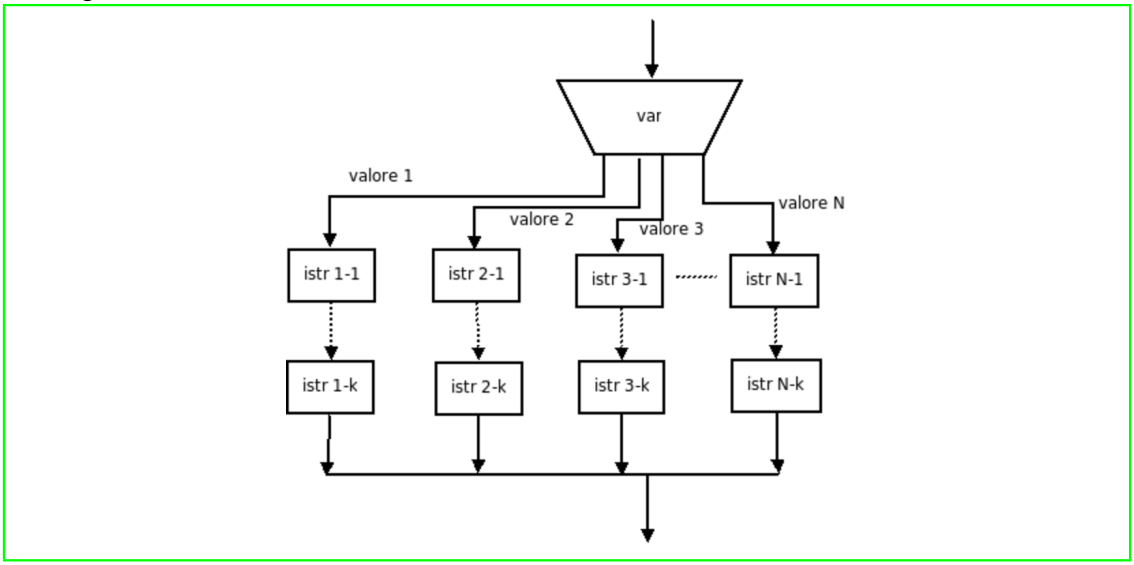
Nel linguaggio di progetto il costrutto da utilizzare è:

```
valuta (var) {
   caso valore1:

      sequenza di istruzioni 1
```

```
    caso valore2:

       sequenza di istruzioni 2
    .
    .
    .
    caso valoreN:

       sequenza di istruzioni N

    default:

       sequenza di istruzioni N+1

}
```

Nei diagrammi di flusso si può utilizzare il blocco mostrato nella figura 2.27.

Figura 2.27



Il costrutto di selezione multipla non è essenziale perché al suo posto si possono usare opportune combinazioni di selezioni «normali».

Come esempio di uso di questa struttura derivata vediamo la realizzazione di un semplice '**menu di scelta**' che esegue istruzioni diverse in base al valore scelto dall'utente.

```
{
   intero scelta;
   esegui {
      stampa ("Menu delle scelte");
      stampa ("1 - Funzione 1");
      stampa ("2 - Funzione 2");
      stampa ("3 - Funzione 3");
      stampa ("9 - Fine");
      leggi (scelta);
      valuta (scelta) {
         caso 1:
```

```
            stampa ("Hai scelto la funzione 1");
            salta;
         caso 2:
            stampa ("Hai scelto la funzione 2");
            salta;
         caso 3:
            stampa ("Hai scelto la funzione 3");
            salta;
         caso 9:
            stampa ("Arrivederci e grazie!");
            salta;
         default:
            stampa ("Scelta errata");
   }
   mentre (scelta != 9);
}
```

L'istruzione '**salta**' porta il flusso di esecuzione al termine del costrutto '**valuta**' e ha lo scopo di non far eseguire le istruzioni dei casi successivi.

Il caso '**default**' viene inserito per eseguire un blocco di istruzioni nel caso il valore della variabile valutata non sia nessuno di quelli previsti.

Ovviamente l'esempio non ha alcuna utilità pratica e serve solo a mostrare le modalità di impostazione e gestione del menu.

2.4.2 Iterazione calcolata

L'iterazione calcolata è solo un modo più compatto di scrivere il ciclo con controllo in testa; per tale motivo nei diagrammi di flusso non esiste alcun costrutto particolare che la rappresenti.

Nel linguaggio di progetto abbiamo invece:

```
per (cont=valore1;cont<valore2;cont=cont+1) {

      sequenza di istruzioni

}
```

Il significato dell'istruzione è abbastanza intuibile:

- ***cont*** viene inizializzato al ***valore1***;
- il ciclo viene eseguito finché rimane vera la condizione che ***cont*** sia minore di ***valore2***;
- ad ogni passo del ciclo ***cont*** viene incrementato di una unità.

Rispetto all'iterazione con controllo in testa vista precedentemente si notano:

- lo spostamento dell'istruzione di inizializzazione del contatore, che prima avveniva fuori dal costrutto del ciclo e adesso è inglobata nell'intestazione;
- lo spostamento dell'istruzione di incremento del contatore, che prima avveniva nel corpo del ciclo e adesso è inglobata nell'intestazione.

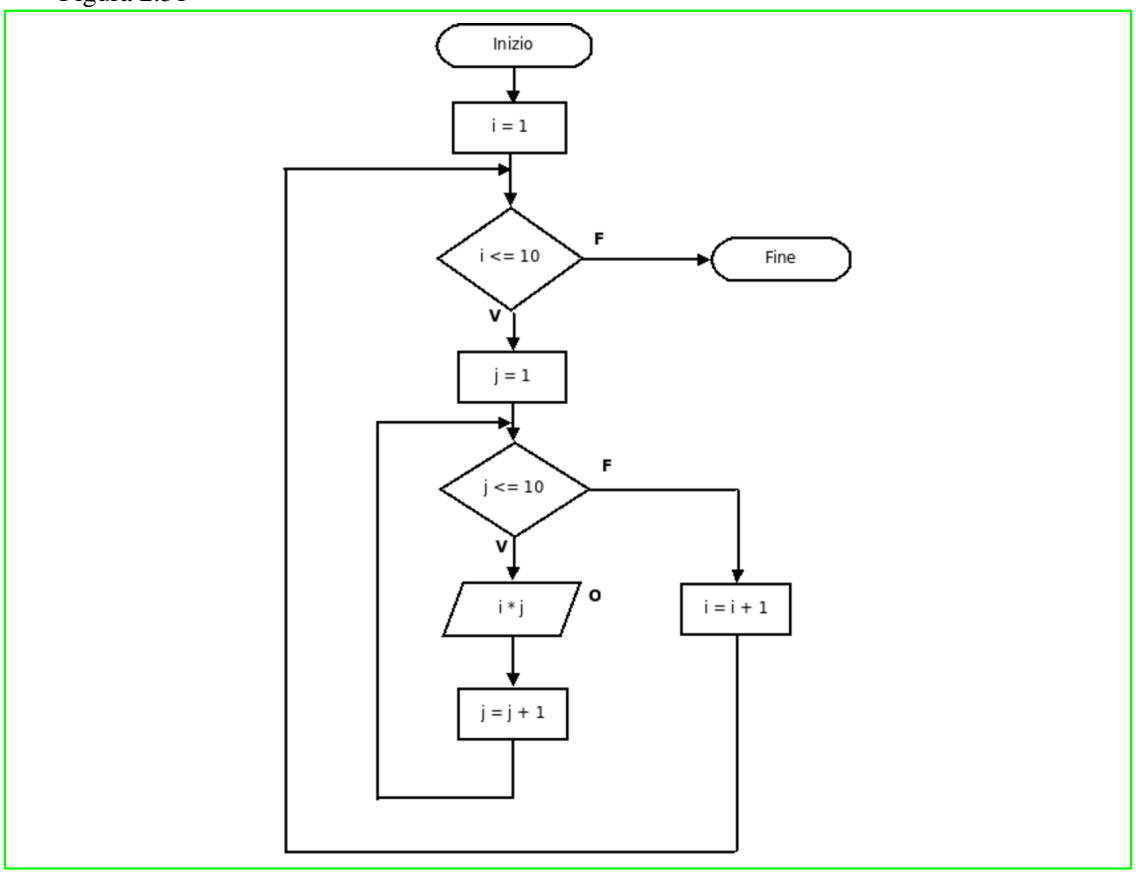
Quello mostrato può essere considerato solo un esempio di ciclo calcolato: nulla vieta infatti di avere valori iniziali, condizioni e incrementi del tutto diversi.

Come esempio vediamo l'algoritmo per la stampa della tabellina pitagoriga per i valori da 1 a 10 dove usiamo due iterazioni calcolate una annidata nell'altra.

```
{
   intero i, j;
   per (i=1;i<=10;i=i+1) {
      per (j=1;j<=10;j=j+1) {
         stampa (i*j);
      }
   }
}
```

Nella figura 2.31 vediamo anche il relativo diagramma di flusso.

Figura 2.31



2.5 La programmazione strutturata

Con il termine '**programmazione strutturata**' si intende una tecnica di realizzazione degli algoritmi e dei relativi programmi che prevede il solo utilizzo delle strutture di controllo fondamentali.

In realtà si può parlare di algoritmi o programmi '**strutturati**' anche in presenza di strutture di controllo derivate, in quanto esse, come abbiamo visto in precedenza, sono ottenibili da quelle fondamentali.

Ogni struttura di controllo può essere immaginata come un blocco, con un solo flusso in entrata ed uno solo in uscita, che al suo interno può contenere una singola istruzione o una ulteriore struttura di controllo (annidamento).

Come esempio consideriamo il problema del calcolo del fattoriale di un numero ***n*** maggiore di 1 e risolviamolo con i tre diversi algoritmi descritti dai diagrammi a blocchi delle figure 2.32, 2.33 e 2.34; nelle tre soluzioni viene tralasciato il controllo sulla validità dell'input.

Figura 2.32

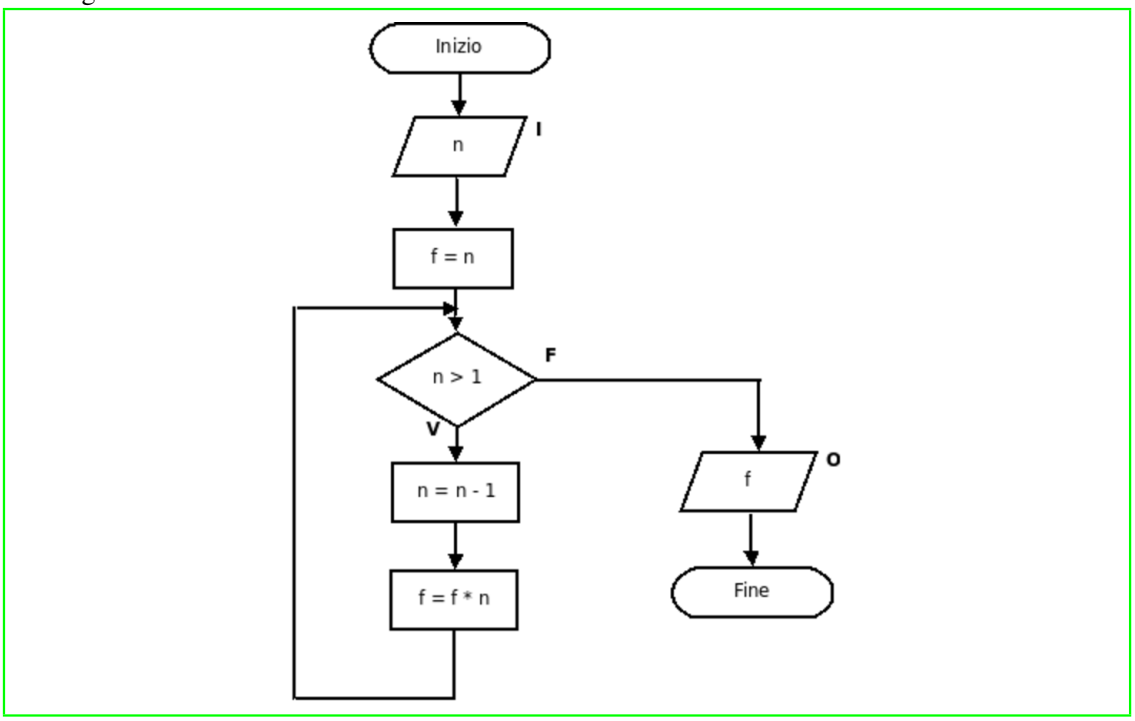


Figura 2.33

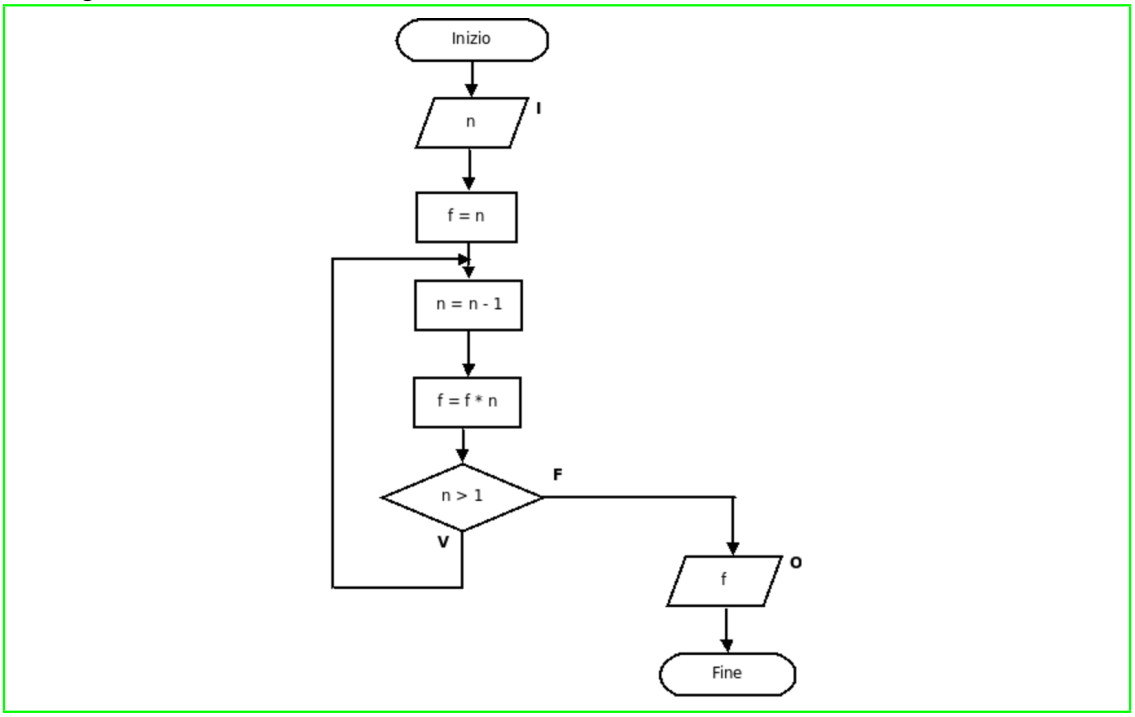
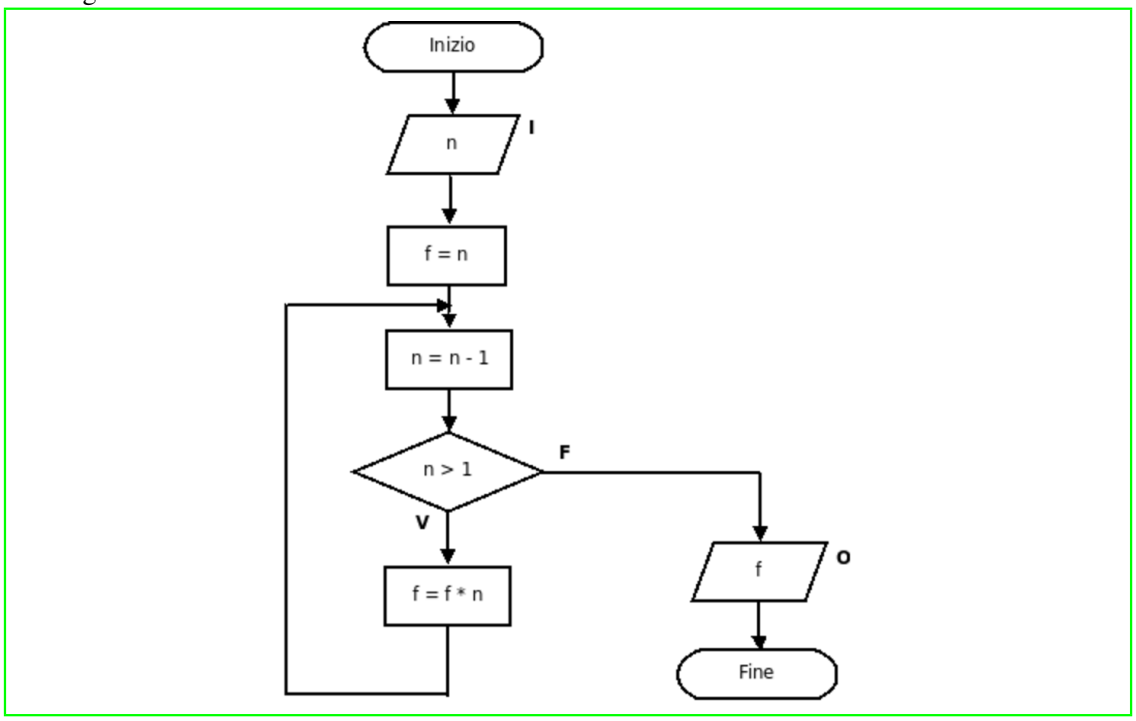


Figura 2.34



Questi tre algoritmi sono '**`funzionalmente equivalenti`**' cioè danno gli stessi risultati partendo dagli stessi dati di input.

Il primo algoritmo contiene un ciclo con controllo in testa, il secondo un ciclo con controllo in coda, il terzo è da scartare perché contiene un ciclo non valido che ha il controllo né in testa né in coda.

2.5.1 Il teorema di Jacopini-Bohm

Ci si potrebbe chiedere se le tre strutture fondamentali formano un insieme di strutture '**`completo`**' (cioè sufficiente a descrivere tutti gli algoritmi).

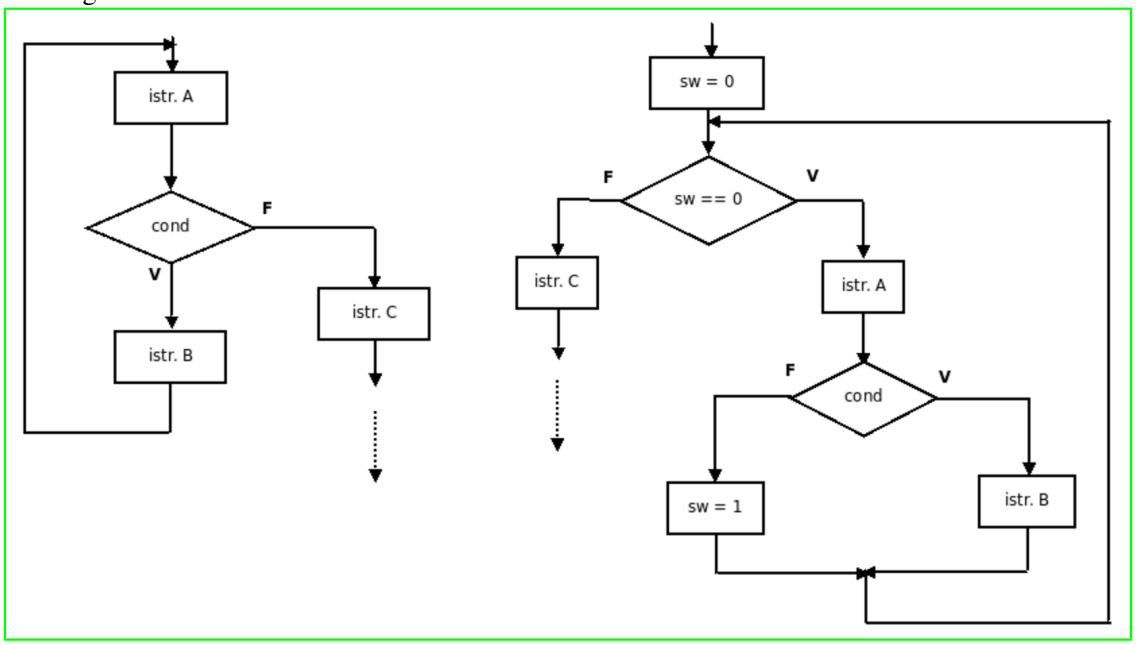
A tale proposito esiste il teorema di '**`Jacopini – Bohm`** ' del 1966 (che non dimostriamo) che afferma: «ogni algoritmo può essere espresso con le sole tre strutture di controllo fondamentali».

Quindi è lecito pretendere da parte dei programmatori l'uso delle tecniche di programmazione strutturata, anche perché con esse si hanno i seguenti vantaggi:

- maggiore facilità di uso della metodologia *top-down* (illustrata nel prossimo paragrafo);
- maggiore leggibilità degli algoritmi;
- maggiore possibilità di individuazione di eventuali errori nella logica risolutiva.

Nella figura 2.35 vediamo: a sinistra una porzione di algoritmo non strutturato (il ciclo ha il controllo né in testa, né in coda), a destra la sua strutturazione grazie all'uso di una variabile di tipo '**`switch`**' (segnalatore), chiamata ***sw***:

Figura 2.35



L'uso di variabili di segnalazione avviene solitamente nel seguente modo:

- la variabile viene «spenta» (assume il valore 0) all'inizio dell'algoritmo;
- viene poi «accesa» (assume il valore 1) al verificarsi di una certa condizione;
- infine viene «testata» in modo da decidere le azioni da intraprendere in base al suo valore.

2.6 Le tecniche Top-Down

Un algoritmo si ottiene come risultato di un procedimento di '**`analisi`**' che può essere suddiviso in tre fasi:

- ' **`definizione del problema`**' con i suoi dati in ingresso ed in uscita;
- individuazione del '**`metodo`**' di soluzione;
- '**`descrizione`**' dell'algoritmo, con il linguaggio di progetto o con il diagramma a blocchi.

Una metodologia di analisi molto usata e anche molto conveniente è quella che fa uso delle tecniche *top-down* o delle '**`scomposizioni successive`**'.

Esse si basano appunto su una scomposizione del problema in sottoproblemi che a loro volta possono essere ulteriormente scomposti fino ad arrivare a sottoproblemi molto semplici costituiti solo da operazioni molto elementari, direttamente «eseguibili» da parte dell'esecutore.

Naturalmente tale metodologia è utile soprattutto in presenza di problemi abbastanza complessi; in caso di problemi semplici, come quelli degli esempi finora esaminati, la sequenza di operazioni proposta nei vari algoritmi risolutivi, può essere considerata già sufficientemente semplice.

Ad ogni sottoproblema viene associato un «sottoalgoritmo» o sottoprogramma, che nei diagrammi di flusso è individuato dal blocco rettangolare con le bande laterali mostrato nella figura 2.2 (tralasciamo invece l'uso dei sottoalgoritmi nel linguaggio di progetto).

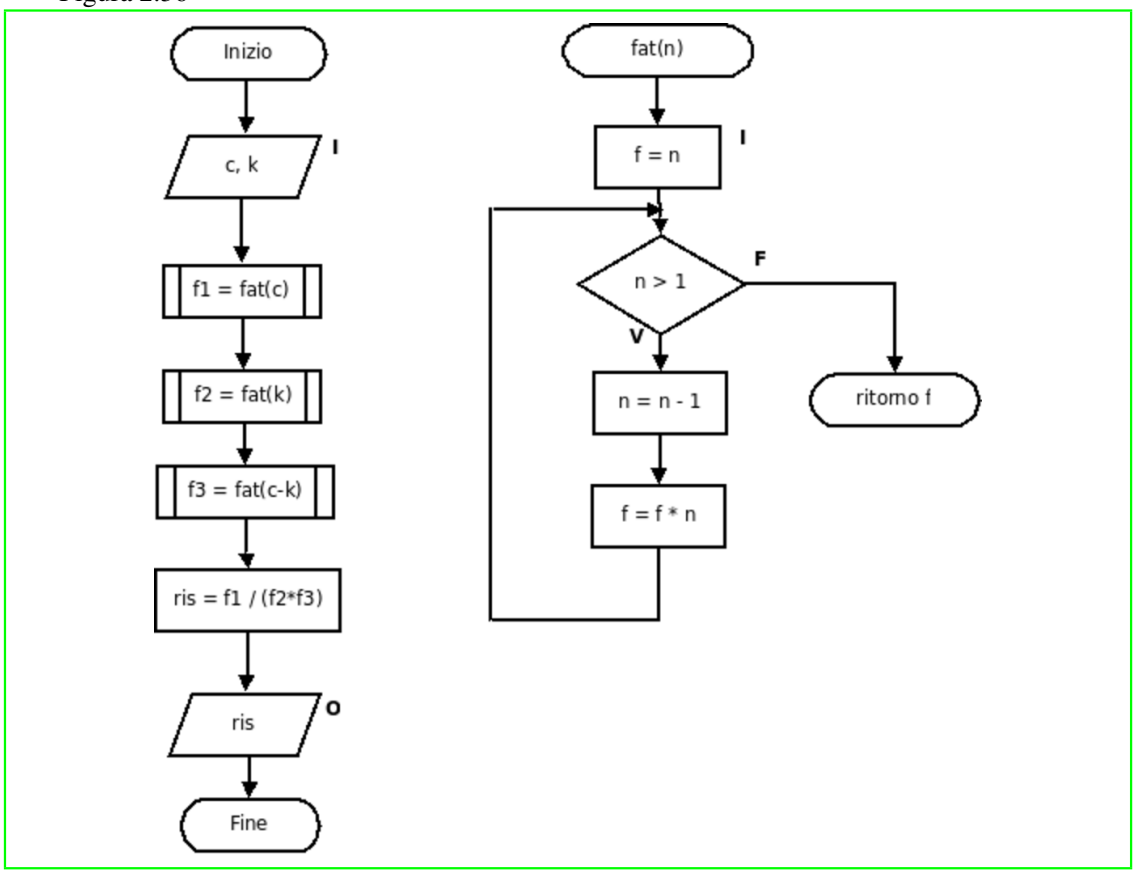
Il dettaglio del sottoalgoritmo viene descritto con i consueti blocchi scrivendo però nel blocco iniziale il suo nome e l'eventuale lista dei valori (parametri) su cui deve operare e nel blocco di fine la parola «ritorno» seguita dall'eventuale valore di ritorno.

Come esempio consideriamo il problema del calcolo del coefficente binomiale «c su k» dove ***c*** e ***k*** sono due interi positivi con c > k.

La soluzione proposta si basa sulla formula di calcolo del coefficente e cioè: c! / (k!*(c-k)!) dove il simbolo «!» indica il fattoriale; nell'algoritmo si fa uso di un opportuno sottoalgoritmo per il calcolo del fattoriale di un valore ***n***.

Il tutto è mostrato nella figura 2.36.

Figura 2.36



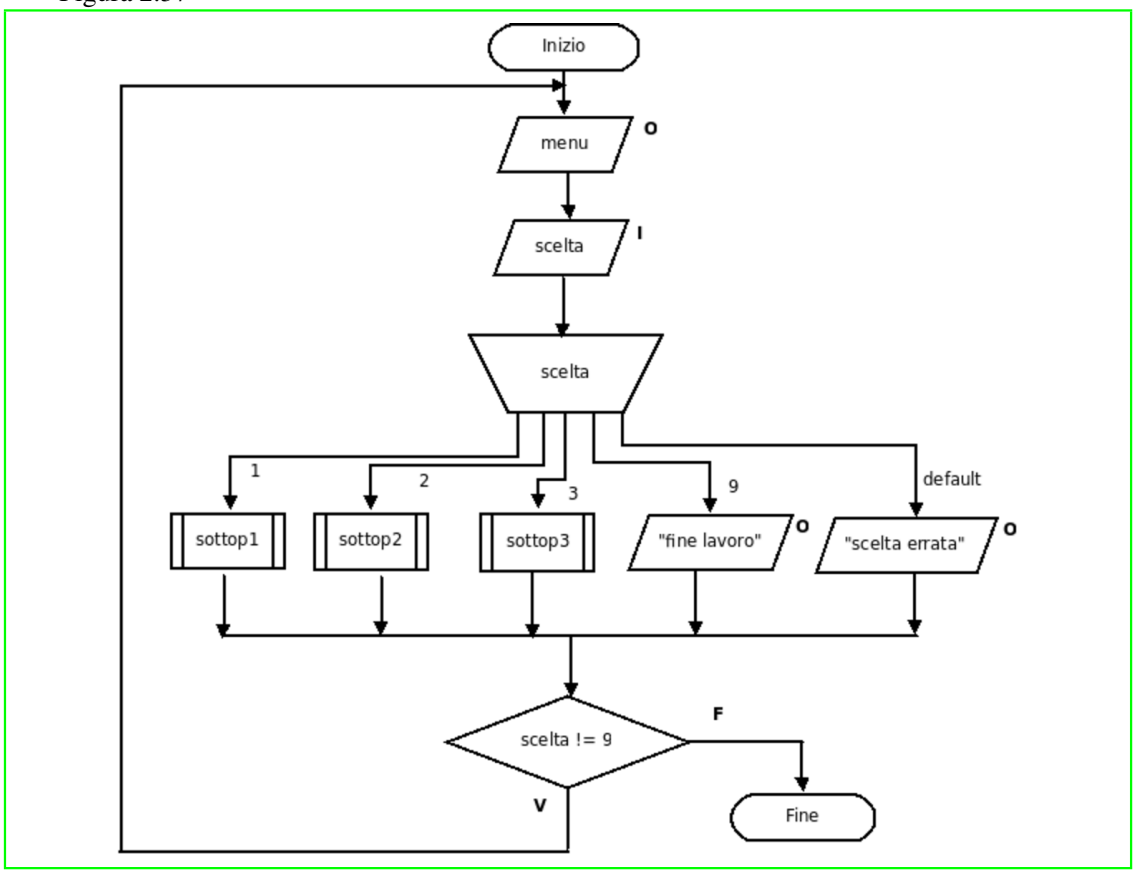
In generale le tecniche di analisi e programmazione top-down sono da consigliare per almeno tre motivi:

- '**`semplificazione`**' dell'algoritmo o del programma che viene scomposto in parti presumibilmente più semplici; occorre però prestare attenzione all'incremento di complessità dovuto all'esigenza di integrare i vari sottoprogrammi;
- '**`risparmio di istruzioni`**' nel programma; nell'esempio precedente appare abbastanza ovvio come, con l'uso del sottoprogramma ***fat***, si evita di scrivere tre volte il procedimento per il calcolo del fattoriale, come sarebbe richiesto dalla formula del coefficente binomiale;
- '**`riuso del codice`**'; si parla in questo caso di '**`programmazione modulare`**' cioè del fatto che si possono scrivere sottoprogrammi (o '**`moduli`**') in modo che possano essere riutilizzati per programmi diversi e da persone diverse.

Questi temi, pur molto interessanti, esulano dagli scopi di questa dispensa e quindi non vengono ulteriormente approfonditi al pari delle nozioni sui parametri e sui valori di ritorno dei sottoprogrammi; per approfondimenti su questi aspetti si rimanda allo studio di specifici linguaggi di programmazione.

Concludiamo con un ultimo esempio, mostrato nella figura 2.37, in cui vediamo il diagramma di flusso relativo alla gestione di un menu che permette di scegliere tra tre opzioni, ognuna corrispondente ad un certo sottoprogramma (non ulteriormante dettagliato).

Figura 2.37



I linguaggi di programmazione

Da completare.

Strutture dati astratte

Da completare.

Documentazione del software

Da completare.