Appunti di Fondamenti di Informatica

Generalità sugli algoritmi

LA NOZIONE DI ALGORITMO

La nozione di **algoritmo** riveste un ruolo di notevole rilievo nei fondamenti della teoria della programmazione dei calcolatori elettronici. Se noi pensiamo ad un libro di aritmetica, esso è una raccolta delle nozioni riguardanti il sistema di numerazione arabo e contiene la descrizione dettagliata dei procedimenti necessari per eseguire le operazioni aritmetiche elementari su numeri rappresentati nel sistema posizionale decimale. Tali procedimenti di calcolo vennero chiamati, da un certo momento in poi, **algoritmi** ed il termine *algoritmo* viene perciò da quel momento utilizzato come sinonimo di "procedimento di calcolo". La necessità di formulare definizioni più precise del concetto di algoritmo ha portato poi allo sviluppo di una vera e propria **teoria degli algoritmi**.

Def. *Si dirà perciò* ***algoritmo*** *un "insieme di istruzioni che definiscono una sequenza di operazioni mediante le quali si risolvono tutti i problemi di una determinata classe".*

Questa definizione viene completata dalla enunciazione delle **proprietà** sottoelencate, che caratterizzano maggiormente la nozione di algoritmo:

1) **carattere finito**: il numero delle istruzioni che definiscono un algoritmo è finito e le operazioni da esse specificate vengono eseguite un numero finito di volte. Naturalmente, affinché un algoritmo possa essere veramente utile, il numero delle istruzioni che lo compongono deve essere limitato, mentre una singola operazione può anche essere eseguita un numero molto alto di volte (questo è, ad esempio, il caso dei **procedimenti iterativi** che, come vedremo, si interrompono solo quando si è verificata una certa condizione);
2) **carattere deterministico**: ogni istruzione che fa parte di un algoritmo provoca l'esecuzione di una o anche più operazioni. Tali operazioni devono essere definite "senza ambiguità", in modo tale, cioè, da garantire che i risultati ottenuti utilizzando più volte un algoritmo per la soluzione di un problema siano sempre gli stessi, a prescindere dalla persona o dalla macchina che esegue l'algoritmo stesso;

3) **carattere di realizzabilità pratica**: tutte le operazioni che vengono richieste dalle istruzioni di un algoritmo devono essere effettivamente eseguibili, cioè devono essere già conosciute in precedenza o dall'utente o dalla macchina che le deve eseguire.

RAPPRESENTAZIONE GRAFICA DEGLI ALGORITMI

Trattiamo adesso un noto algoritmo elementare e descriviamo poi la sua rappresentazione grafica.

*Trovare il **massimo comune divisore** di due numeri dati, m e n, interi positivi.*

RISOLUZIONE E SPIEGAZIONE: Un algoritmo che risolve questa "classe di problemi" viene detto **algoritmo di Euclide**. Fondamentalmente, esso si basa sulla ripetuta applicazione della seguente proprietà: data una qualsiasi divisione tra un dividendo e un divisore, entrambi interi, ogni numero che divide sia il dividendo sia il divisore ne divide anche il resto nel caso che questo non sia nullo. Difatti, dati i due numeri interi positivi m e n, supponendo che m sia maggiore di n, possiamo sempre scrivere che

$$m = q \cdot n + r$$

dove q è il quoziente ed r è il resto della divisione m/n.

Se risulta r=0, allora m è un multiplo di n per cui n è il massimo comun divisore ricercato; invece, se il resto non è nullo, dalla relazione precedente ricaviamo che

$$m - q \cdot n = r$$

Confrontando le due relazioni, si deduce che l'insieme dei divisori comuni di m ed n coincide con l'insieme dei divisori comuni di n ed r: quindi, il massimo comun divisore di m ed n corrisponde al massimo comun divisore di n ed r.

Allora, il nostro problema diventa il seguente: trovare il massimo comun divisore di n ed r , dove r è il resto della divisione di m per n (supponendo che $m \geq n$). Se si applica nuovamente ad n ed r il ragionamento fatto sui numeri m ed n e se si procede in modo "iterativo", si realizza un procedimento che viene cosi schematizzato:

1) $m = q_1 \cdot n + r_1$ con $0 \leq r_1 < n$

   se $r_1 = 0 \rightarrow$ è il Max Comun Divisore

2) se $r_1 \neq 0 \rightarrow$ $n = q_2 \cdot r_1 + r_2$ con $0 \leq r_2 < r_1$

   se $r_2 = 0 \rightarrow$ r_1 è il MCD

3) se $r_2 \neq 0 \rightarrow$ $r_1 = q_3 \cdot r_2 + r_3$ con $0 \leq r_3 < r_2$

   se $r_3 = 0 \rightarrow$ r_2 è il MCD

.....
.....

4) se $r_{i-1} \neq 0 \rightarrow r_{i-2} = q_i \cdot r_{i-1} + r_i$ con $0 \leq r_i < r_{i-1}$

se $r_i = 0 \rightarrow r_{i-1}$ è il MCD

.....

Il procedimento prosegue ancora, ma avrà certamente un termine: difatti i resti r_i costituiscono una successione decrescente di numeri interi, il cui termine di valore massimo è proprio r_i; perciò, il massimo comun divisore cercato è l'ultimo resto della successione r_i diverso da zero. Supponiamo, ad esempio, che si sia arrivati al resto r_5 diverso da zero: allora, in accordo a quanto visto nei precedenti passaggi, si effettua la divisione di r_4 per r_5 e si ottiene che

$$r_4 = q_6 \cdot r_5 + r_6$$

Se, a questo punto, si trova che $r_6=0$, significa che r_4 è un multiplo di r_5, per cui il massimo comun divisore è proprio r_5, cioè l'ultimo resto ottenuto prima di avere resto zero (cioè r_6).

Formalizzando l'esposizione di tale algoritmo secondo la definizione riportata nel paragrafo precedente, otteniamo il seguente **elenco di istruzioni**:

Istruzione 1 - (*determinazione del maggiore*) Una volta che sono stati introdotti i due numeri interi, si indichi con m il numero maggiore e con n il numero minore;

Istruzione 2 - (*calcolo del resto r*) Si divida il maggiore dei due numeri (cioè m) per il minore (n), e si indichi il resto della divisione per r;

Istruzione 3 - (*confronto di r con zero*) Si confronti il valore del resto r con zero. Se si trova r=0, l'algoritmo può terminare ed n è il massimo comun divisore ricercato; se r non è zero, si esegua l'istruzione successiva;

Istruzione 4 - (*sostituzione dei numeri dati*) Sostituire alla coppia di valori m,n la coppia n,r (dove sicuramente n è maggiore di r) e ritornare ad eseguire l'istruzione 2 utilizzando questi nuovi valori;

All'inizio della descrizione di ogni istruzione, c'è una frase racchiusa tra parentesi: essa descrive, in modo quanto più conciso possibile, quello che deve essere fatto. Queste frasi sono poi seguite da una descrizione più dettagliata della o delle varie operazioni che devono essere effettuate prima di poter considerare eseguita l'istruzione nella quale sono contenute.

Naturalmente, *tutte le istruzioni, a meno di indicazioni contrarie, devono essere eseguite in modo sequenziale a partire da quella indicata con il numero più basso verso quella di numero superiore.* Per esempio, nel caso dell'algoritmo di prima, una volta eseguite le istruzioni 1, 2 e 3, si possono presentare due diverse situazioni: se il resto è nullo allora il problema è risolto e l'algoritmo si può considerare concluso; se invece il resto non è nullo, allora si passa ad eseguire l'istruzione 4 che, a sua volta, rimanda all'istruzione 2. Questo procedimento prosegue finché, come risultato della istruzione 3, si ottiene r=0.

Diagrammi di flusso

La necessità di descrivere dettagliatamente la **sequenza operativa completa** di un algoritmo e quella, ancora più evidente, di dare a tale descrizione caratteristiche di chiarezza e di brevità, hanno portato alla ricerca di metodi convenienti per la descrizione degli algoritmi. Uno dei metodi più utili consiste nella rappresentazione dell'algoritmo mediante un **diagramma**, nel quale *ad ogni istruzione viene associato un simbolo grafico particolare (la cui forma dipende appunto dal tipo di istruzione).* I diagrammi utilizzati per la descrizione degli algoritmi vengono comunemente detti **diagrammi logici** o **diagrammi a blocchi** o **flow-chart** o **diagrammi di flusso**.

I vari **simboli grafici** che costituiscono i diagrammi a blocchi si susseguono dall'alto verso il basso e sono collegati tra loro per mezzo di **tratti rettilinei**; su tali tratti rettilinei è anche stabilito un **orientamento** (mediante frecce) per indicare l'ordine di esecuzione delle istruzioni. Quando la successione di due istruzioni è implicita, allora le frecce possono essere anche omesse.

Per comprendere come sia fatto un generico diagramma di flusso, cominciamo con un esempio concreto e, in particolare, proviamo a rappresentare il diagramma dell'algoritmo di Euclide. Prima di tracciare il diagramma, però, è necessario fare alcune considerazioni preliminari.

Dobbiamo in primo luogo osservare che è indispensabile premettere delle nuove istruzioni alle 4 date in precedenza:

- ad esempio dire, nella istruzione 1, "considerare i due numeri dati" presuppone una operazione mediante la quale i due numeri (cioè i dati su cui si deve operare) vengono introdotti nell'ambiente in cui l'algoritmo viene eseguito; dobbiamo dunque inserire, prima della istruzione 1, una istruzione che indichi l’**introduzione dei dati**;
- allo stesso modo, è necessario aggiungere, prima della fine dell'algoritmo, una istruzione che evidenzi l'**uscita del risultato** dall'ambiente in cui l'algoritmo viene eseguito;
- infine, sarebbe opportuno inserire, ma non lo faremo per brevità, prima che comincino le operazioni, anche una istruzione che permetta all'esecutore dell'algoritmo (sia esso una persona o un calcolatore) di controllare che i dati ricevuti soddisfino le condizioni richieste (cioè che i due numeri siano interi e positivi).

Osserviamo inoltre che l'istruzione 4 avrebbe potuto essere indicata in modo più preciso: infatti, sostituire alla coppia m,n la coppia n,r significa "assegnare" alla variabile indicata con m il valore che aveva in precedenza la variabile n e assegnare alla variabile indicata con n il valore della variabile r. Per indicare una assegnazione di questo tipo si usa una notazione del tipo

<espressione> → <variabile>

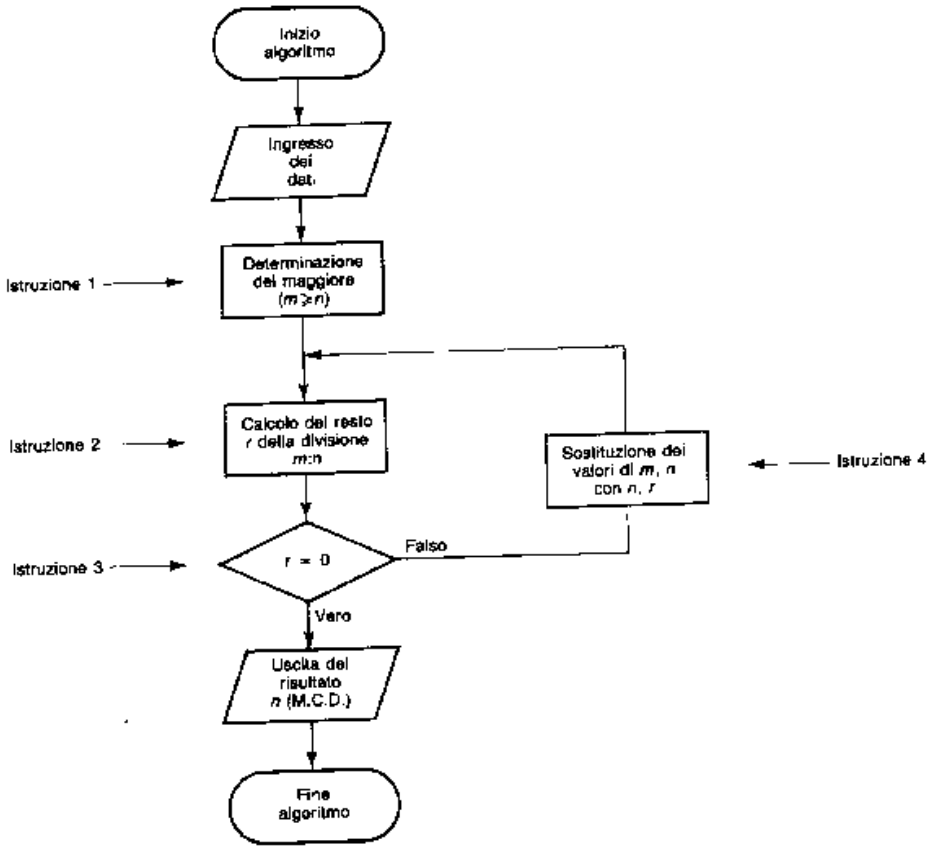
Questa notazione specifica che il valore che si ottiene dal calcolo dell'espressione alla sinistra della freccia deve essere assegnato alla variabile indicata a destra. Ad esempio n → m si legge "n va in m" .

Un altro tipo di notazione, utilizzato in alcuni linguaggi di programmazione, è il seguente:

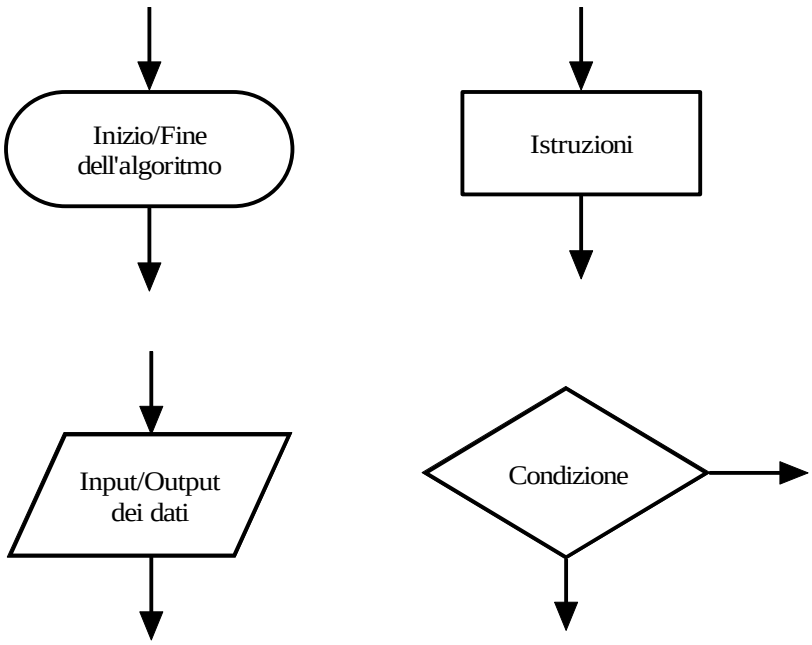
<variabile> := <espressione>

L'uso del simbolo ":=" al posto del semplice "=" ha lo scopo di evidenziare che si tratta di una assegnazione e non di una relazione di uguaglianza.

Sulla base di queste considerazioni, possiamo adesso tracciare il diagramma a blocchi dell'algoritmo di Euclide:



Si nota, in questo diagramma, la presenza dei seguenti simboli grafici (o **blocchi**):



Come si vede, ognuno di essi corrisponde ad una precisa tipologia di istruzioni: c'è un blocco corrispondente all'inizio o alla fine dell'algoritmo, c'è un blocco corrispondente all'esecuzione di

istruzioni generiche (come ad esempio operazioni algebriche tra variabili), c’è un blocco per l’ingresso e l’uscita dei dati ed infine c’è un blocco per la verifica (VERO o FALSO) delle condizioni booleane. Mentre i primi tre blocchi citati presentano un solo ingresso ed una sola entrata, il blocco rappresentato dal rombo prevede un solo ingresso e due uscite. Il rombo viene chiamato **simbolo di decisione** e serve per rappresentare la scelta da effettuare dopo l'esame di una proposizione (in quel caso r=0) che può essere "vera" o "falsa". Perciò *l'algoritmo può proseguire lungo percorsi alternativi a seconda che la proposizione considerata sia vera o falsa.*

ESEMPI DI ALGORITMI NUMERICI

Gli algoritmi che utilizzano, durante la loro esecuzione, le operazioni aritmetiche fondamentali vengono chiamati **algoritmi numerici**; vediamone un esempio molto semplice:

Costruire un algoritmo per la risoluzione dell'equazione di 2° grado $ax^2+bx+c=0$

RISOLUZIONE E SPIEGAZIONE: I dati in ingresso a tale algoritmo sono costituiti dai valori dei coefficienti a,b e c, mentre i dati in uscita saranno evidentemente i valori delle radici dell'equazione. La sequenza di istruzioni che costituisce l'algoritmo richiesto può essere la seguente:

Istruzione 1 - (*introduzione di a, b e c*)

Istruzione 2 - (*a=0 AND b=0*)

Si valuta la proposizione **a=0 AND b=0** nella quale il simbolo **AND** ha la funzione di congiunzione logica (e): questa proposizione risulta cioè vera se i due coefficienti sono entrambi nulli: in questo caso l'equazione da risolvere è degenere (c=0) ed essa non ammette soluzioni qualunque sia c. Perciò, se la proposizione risulta vera si passa all'istruzione 3 (*fine algoritmo*) altrimenti si passa all'istruzione 4

Istruzione 3 - (*segnalazione che l'equazione è degenere e arresto*)

A questo punto l'algoritmo segnala che si tratta di una equazione degenere e si arresta

Istruzione 4 - (*a=0*)

Si valuta la proposizione **a=0**: se la proposizione è vera, tenendo presente che non può essere b=0 (altrimenti l'algoritmo si sarebbe fermato alla istruzione 3), si passa all'istruzione 5, altrimenti si passa alla 8.

Istruzione 5 - (*segnalazione che l'equazione è di primo grado*)

Dato che, con l'istruzione 4, si è accertato che a=0, si segnala che l'equazione è di primo grado e si passa all'istruzione successiva.

Istruzione 6 - (*determinazione della radice per equazione di primo grado*)

Se l'equazione è di 1° grado, l'unica radice si ottiene semplicemente come **x = - c/b**

Istruzione 7 - (*estrazione di x e arresto*)

Si segnala la radice trovata e l'algoritmo si ferma

Istruzione 8 - (*determinazione del discriminante*)

Questa istruzione è valida nel caso, nell'istruzione 4, si sia trovato che a≠0: in questo caso, l'equazione è di 2° grado e bisogna quindi calcolare il discriminante (=b^2-4ac) per stabilire il tipo di radici (reali e distinte, reali e coincidenti oppure complesse coniugate)

Istruzione 9 - (*discriminante ³0*)

Si valuta la proposizione per cui il discriminante risulta positivo o al più nullo: se la proposizione è vera (e si hanno perciò comunque 2 radici reali, distinte o coincidenti) si passa all'istruzione 10, altrimenti si passa all'istruzione 15 (per le radici complesse)

Istruzione 10 - (*discriminate =0*)

Si valuta la proposizione per cui il discriminante risulta =0. Se la proposizione è vera (2 radici reali coincidenti) si passa all'istruzione 11, altrimenti si passa alla 12

Istruzione 11 - (*determinazione delle radici reali coincidenti*)

Si calcola il valore delle due radici reali e coincidenti mediante la semplice formula $x_1 = x_2 = -\frac{b}{2a}$ e si passa all'istruzione 14

Istruzione 12 - (*calcolo della radice quadrata S del discriminante*)

Istruzione 13 - (*determinazione delle radici reali e distinte*)

Si calcolano i valori delle due radici reali e distinte mediante le formule $x_1 = \frac{-b+S}{2a}$ e $x_2 = \frac{-b-S}{2a}$

Istruzione 14 - (*estrazione di x_1 e x_2 ed arresto*)

Dopo aver provveduto al calcolo delle radici o nel caso del discriminante nullo (istruzione 11) o in quello del discriminante positivo (istruzione 13), si segnalano le radici e si arresta il procedimento

Istruzione 15 - (*calcolo della radice quadrata del valore assoluto del discriminante*)

Se il discriminate è minore di zero, le radici sono complesse e coniugate per cui bisogna calcolarne la parte reale e quella immaginaria

Istruzione 16 - (*calcolo della parte reale PR delle radici complesse coniugate*)

Se il discriminante è minore di zero, una volta calcolata la radice quadrata del suo valore assoluto (istruzione 15) si calcola la parte reale delle radici complesse coniugate mediante la semplice formula $PR = \frac{-b}{2a}$

Istruzione 17 - (*calcolo del coefficiente CI della parte immaginaria*)

Si calcola il coefficiente della parte immaginaria mediante $CI = \frac{4ac - b^2}{2a}$

Istruzione 18 - (*estrazione di PR e CI ed arresto*)

Dopo aver provveduto alla estrazione della parte reale e del coefficiente della parte immaginaria delle due radici complesse coniugate, l'algoritmo si arresta.

LA STRUTTURAZIONE DEI PROGRAMMI

Nella costruzione degli algoritmi visti in precedenza non è stata seguita alcuna particolare **disciplina** e questo allo scopo di rendere la loro descrizione il più naturale possibile. Tuttavia, è evidente la necessità che gli algoritmi e quindi i programmi per calcolatore vengano costruiti applicando dei metodi che garantiscano una buona **strutturazione**, che conducano cioè ad algoritmi chiari, il più possibile semplici e comprensibili anche a distanza di tempo dal momento in cui sono stati usati per la prima volta. Tra i metodi proposti, occupa un posto di rilievo la **programmazione strutturata** della quale forniamo adesso qualche cenno.

Con il termine **programmazione strutturata** si intende un metodo di lavoro caratterizzato da due regole principali:

- in primo luogo, la costruzione degli algoritmi deve avvenire in modo **top-down**, ossia per raffinamenti successivi;
- in secondo luogo, è necessario un uso disciplinato delle cosiddette **strutture di controllo**, cioè delle strutture che determinano la successione delle operazione descritte in un diagramma di flusso.

Il procedimento "top-down" viene realizzato descrivendo in modo preciso e formale i diversi livelli di dettaglio a cui si arriva a partire dalle linee generali dell'algoritmo risolutivo; questo permette di raggiungere una descrizione molto dettagliata dell'algoritmo stesso al punto da poter essere immediatamente tradotta in uno specifico **linguaggio di programmazione**. Eseguendo tale impostazione in modo corretto, si riducono i problemi legati alla comunicazione tra le varie parti del programma, si riducono i problemi di **prova** del programma stesso e quelli legati alla cosiddetta **manutenzione**, ossia l'insieme degli interventi rivolti alla correzione, all'aggiornamento o all'eventuale miglioramento dei programmi dopo la loro prima utilizzazione. E' chiaro comunque che il significato di una corretta impostazione top-down dei programmi può essere illustrato con efficacia soltanto se si fa riferimento a programmi di notevole complessità.

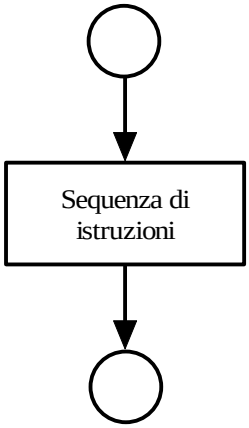
Passiamo adesso alle **strutture di controllo**: l'uso disciplinato di tali strutture è basato su un particolare teorema (di **Jacopini-Bohm**) in cui si afferma *che ogni diagramma di flusso*

può essere trasformato in un diagramma equivalente che utilizza soltanto tre "strutture di controllo fondamentali". Queste tre **strutture fondamentali** sono:

- la **sequenza**, che è la più elementare;
- la **diramazione**, che può essere *semplice* oppure *multipla*;
- l' **iterazione**, la quale a sua volta si divide in *iterazione a condizione finale*, *iterazione a condizione iniziale* e *iterazione a contatore*.

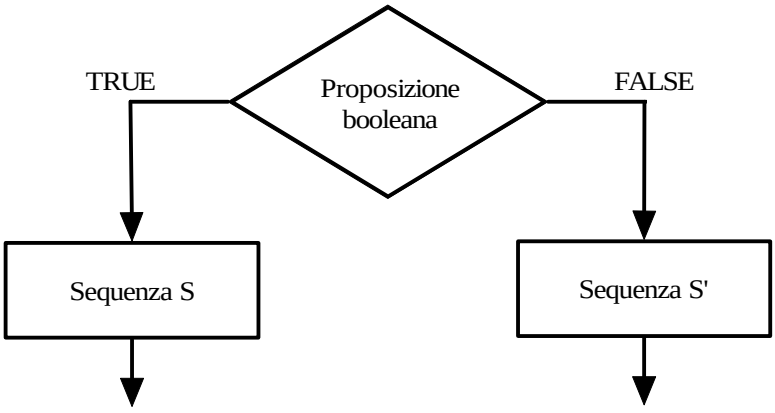
Descriviamo, nell'ordine, queste strutture.

La **sequenza** ha bisogno di ben pochi commenti, in quanto presenta un semplice blocco di operazioni da eseguire una dopo l'altra in modo, appunto, sequenziale. Il corrispondente diagramma a blocchi è dunque banale:



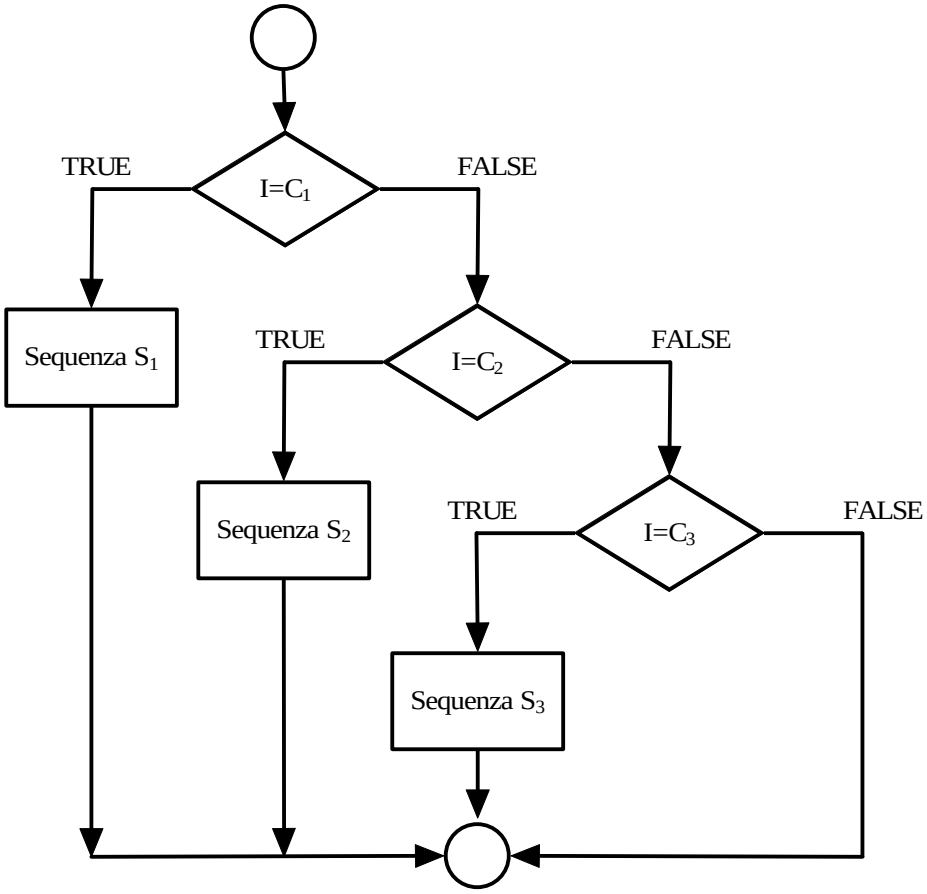
In questa figura, con i "cerchietti" abbiamo indicato dei *nodi generici* del diagramma in cui la sequenza in esame è inserita.

Per quanto riguarda la **diramazione**, abbiamo una **proposizione booleana** che può assumere o il valore TRUE oppure il valore FALSE: allora, nel caso di valore TRUE deve essere eseguita una certa sequenza S, nel caso di valore FALSE deve essere eseguita un'altra sequenza S'. Tanto per avere un esempio concreto, nel linguaggio Pascal la diramazione corrisponde al ciclo IF THEN ELSE. Il diagramma a blocchi è ovviamente il seguente:

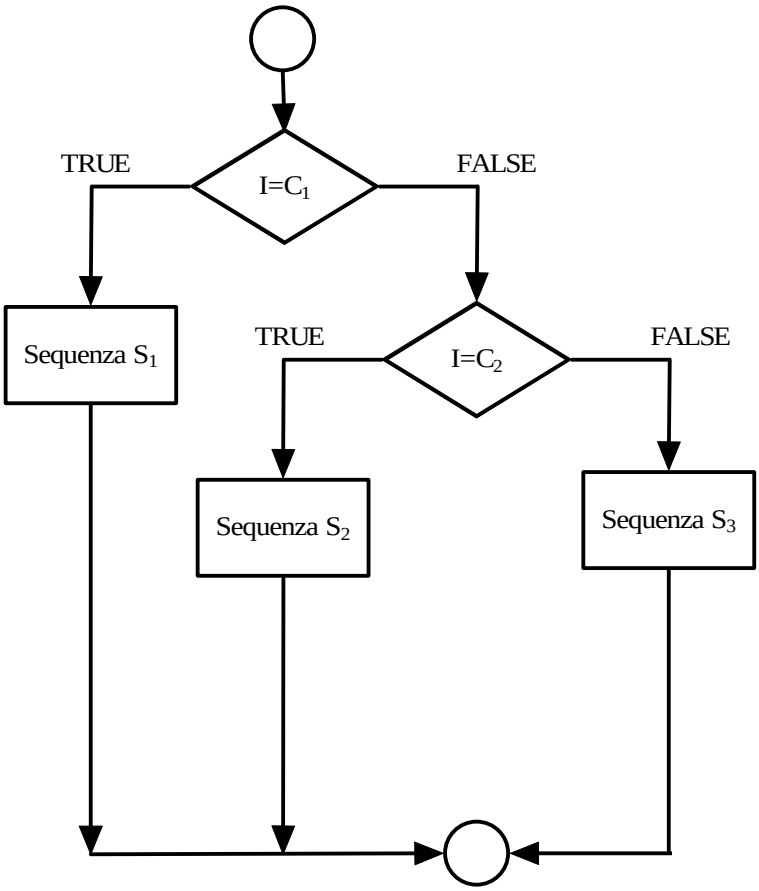


Una struttura di controllo derivata dalla diramazione è la cosiddetta **diramazione multipla** (corrispondente alla CASE nel Pascal): in questo caso, viene esaminata per prima cosa una variabile I che si suppone possa assumere i valori C1,C2,..,Cn; ad ognuno di questi valori corrisponderà una certa sequenza, per cui se il valore di I è C1 si eseguirà la sequenza S1,..., se il valore di I è C2 si eseguirà la sequenza S2,..., se il valore di I è Cn si eseguirà la sequenza Sn.

Volendo rappresentare la diramazione multipla tramite un diagramma a blocchi, subentra una complicazione, derivante dal fatto che, al contrario della diramazione semplice, non esiste un blocco convenzionale (come il rombo) corrispondente alla diramazione multipla. Di conseguenza, *il diagramma di flusso della diramazione multipla può essere visto come una combinazione in cascata di diramazioni semplici.* Per chiarire questo concetto, supponiamo che la diramazione multipla preveda, per la variabile I, tre soli valori C_1 e C_2, cui corrispondono perciò tre distinte sequenze S_1, S_2 ed S_3. Il diagramma a blocchi, in questo caso, può essere il seguente:



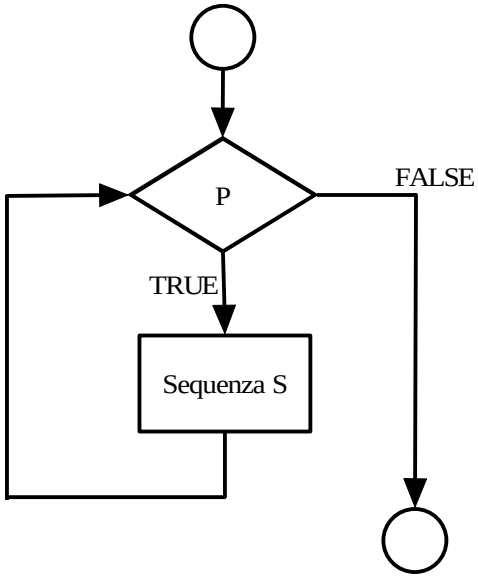
In base a questa rappresentazione, viene per prima cosa controllata la condizione $I=C_1$: se risulta vera, allora viene eseguita la sequenza di istruzioni S_1 corrispondente, altrimenti si passa a controllare la condizione $I=C_2$; anche in questo caso, se essa risulta vera viene eseguita la sequenza di istruzione S_2 corrispondente; se invece la condizione $I=C_2$ risulta falsa, allora le possibilità sono due: quella riportata nella figura presuppone che la variabile I possa assumere anche valori diversi da C_1,C_2 e C_3, il che ovviamente rende necessario l'uso del blocco di decisione relativo alla condizione $I=C_3$; se invece l'algoritmo considerato fosse tale che I debba necessariamente assumere uno tra i valori C_1,C_2 e C_3, allora il blocco di decisione relativo alla condizione $I=C_3$ non sarebbe necessario: infatti, se I non assume né il valore C_1 (prima condizione) né il valore C_2 (seconda condizione), sicuramente dovrà essere pari a C_3. Se facciamo dunque l'ipotesi che I debba necessariamente assumere uno tra i valori C_1,C_2 e C_3, allora il diagramma a blocchi si può semplificare nel modo seguente:



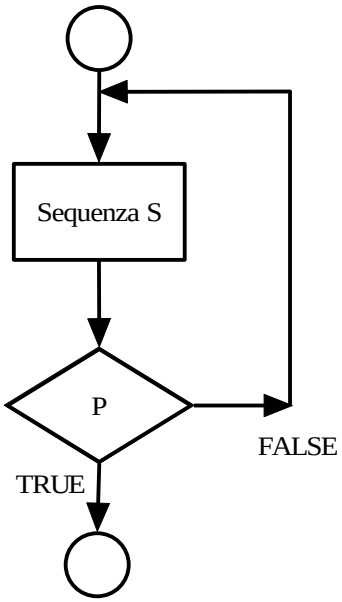
Evidentemente, una volta appurato che la condizione $I=C_2$ è anch'essa falsa, si può subito passare all'esecuzione della sequenza S_3, dato che sicuramente è verificata la condizione $I=C_3$.

Tornando adesso alle strutture fondamentali di controllo, consideriamo l' **iterazione**, che può essere suddivisa in varie categorie:

- c'è l'**iterazione a condizione iniziale**, nella quale si esegue (DO) o meno la sequenza S mentre (WHILE) una certa proposizione booleana P risulta vera (TRUE). Naturalmente, la proposizione P può risultare falsa alla prima esecuzione: in questo caso la sequenza S non è eseguita alcuna volta e l'algoritmo prosegue; il diagramma a blocchi è dunque il seguente:



- c'è poi l'**iterazione a condizione finale** (REPEAT) con la quale la sequenza S viene eseguita sempre una prima volta (a differenza di quanto succede nella WHILE) e poi viene ripetuta fino a quando (UNTIL) la proposizione P diventa vera (TRUE). Una volta diventata vera, l'algoritmo prosegue. Il diagramma a blocchi è dunque il seguente:



- infine c'è l'**iterazione a contatore** (FOR), nella quale la sequenza S viene eseguita se la cosiddetta **variabile contatore I**, che parte da un valore iniziale E e viene incrementata di un passo (STEP) dopo ogni esecuzione, non supera il valore finale E'. La sequenza non viene più eseguita quando il contatore supera il valore E'. Ciò può verificarsi anche alla prima esecuzione, caso in cui la sequenza non viene eseguita nemmeno una volta.

Per finire, c’è un'ultima struttura, chiamata **LOOP-EXIT**, che non è tuttavia disponibile su tutti i linguaggi di programmazione (ad esempio non è disponibile in Pascal): essa prevede la possibilità di uscire da un punto intermedio di un ciclo iterativo al verificarsi di una determinata condizione.

Autore: **SANDRO PETRIZZELLI**
e-mail: sandry@iol.it
sito personale: http://users.iol.it/sandry
succursale: http://digilander.iol.it/sandry1