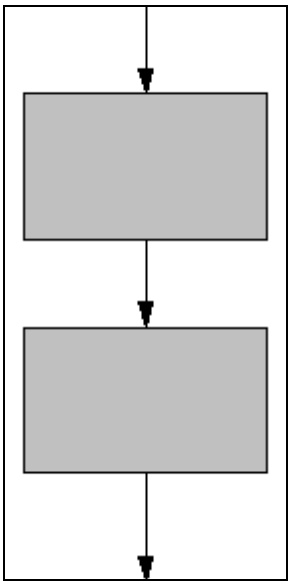
Richiami di programmazione strutturata

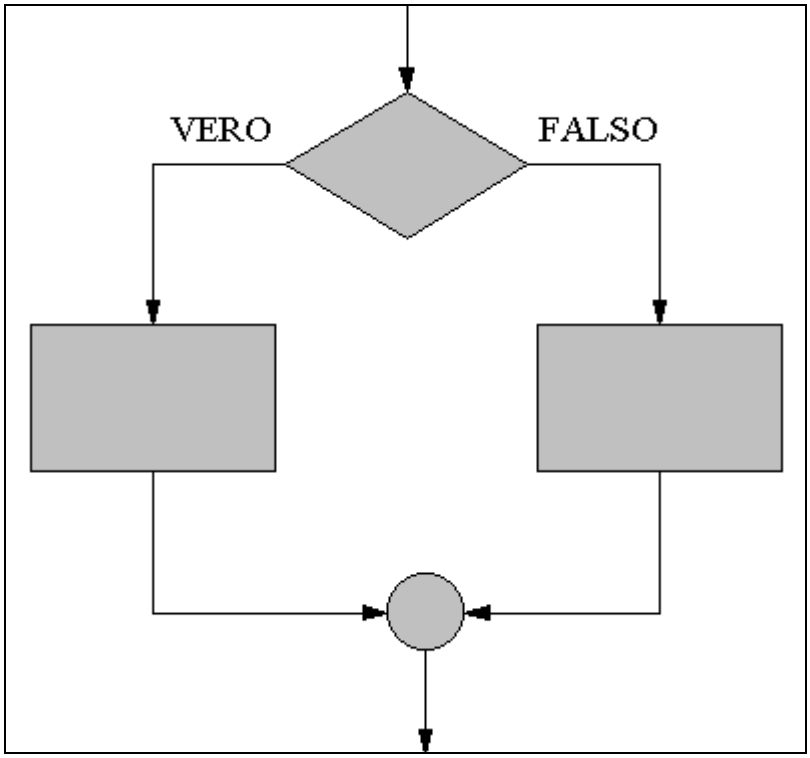
Costruire un programma senza un *metodo* significa affrontare con un unico grosso sforzo la complessità del problema. Si rischia di perdere correttezza e modificabilità del programma. Con "metodo" si intende un insieme di regole generali e flessibili tendenti ad indirizzare le attività di analisi al raggiungimento dell'obiettivo finale attraverso un processo di sviluppo a fasi successive.
Il primo passo da realizzare è un'analisi preventiva del problema. Tale analisi si chiama *sviluppo top-down*: esso procede partendo dal livello più elevato e, con esplosioni successive, scendendo gradualmente fino al livello di dettaglio desiderato.
A questo punto si passa al secondo passo, ossia si procede cercando di suddividere il progetto in moduli che svolgano completamente una funzione e solo quella. Tali moduli devono essere il più possibile indipendenti nel senso di dipendere dagli altri moduli solo in termini di dati e non di condizioni.
A questo punto si è operata la cosiddetta *programmazione strutturata*, ossia si è realizzato un programma con le qualità di correttezza e di modificabilità, suddiviso in moduli indipendenti che scambiano solo dati.
Le strutture logiche di controllo fondamentali per la programmazione strutturata sono individuate in: *sequenza*, *selezione* e *iterazione*.

La **sequenza** consente di individuare le varie operazioni che devono essere eseguite e l'ordine di tali operazioni. In tal modo è possibile costruire il flusso di operazioni che portano ad una certa informazione.



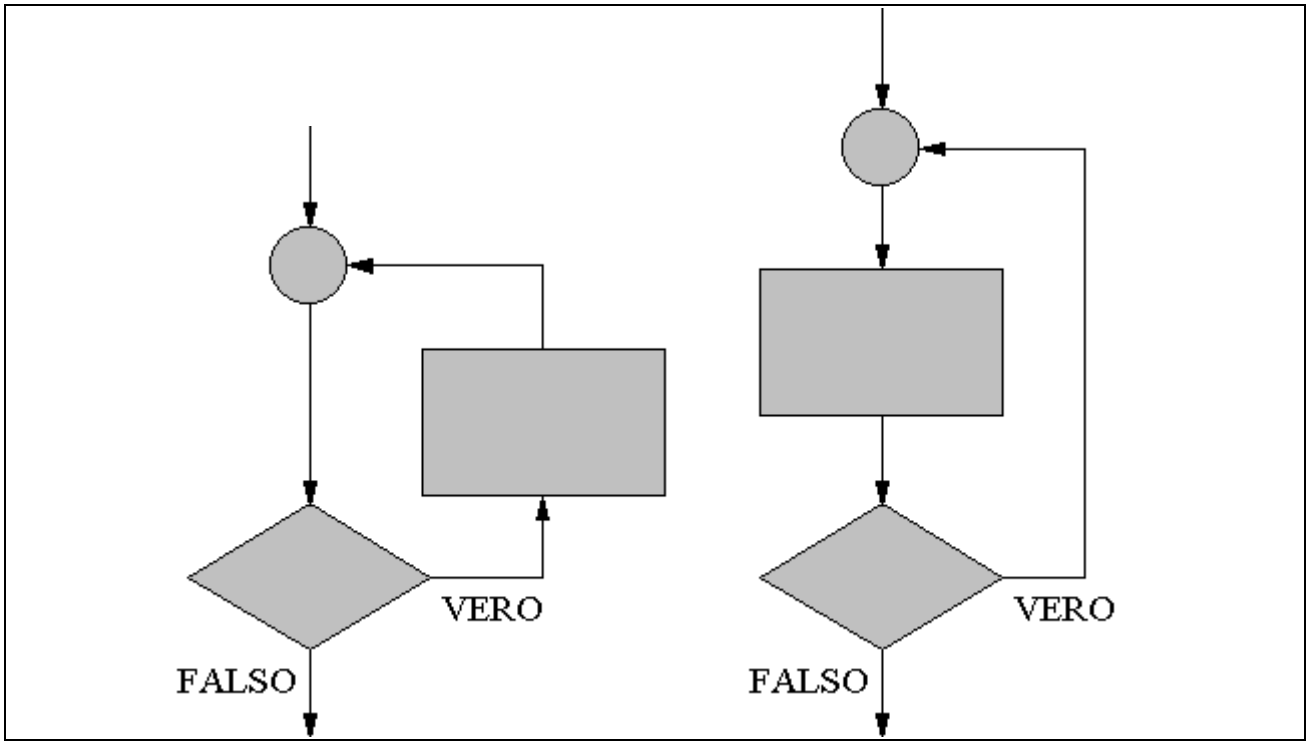
Sequenza

La **selezione** consente di stabilire azioni alternative dipendenti dal fatto di avverarsi o meno di una certa condizione.



Selezione

Infine, **l'iterazione** consente di ripetere un certo numero di volte una certa operazione finché è vera una certa condizione.



Iterazioni

Il metodo della programmazione strutturata porta automaticamente a programmi senza GOTO, ossia senza salti che ne aumentano la complessità, la possibilità di errori, la difficoltà di interpretazione e manutenzione.
Per realizzare tale strutture logiche vi sono diversi metodi ma essenzialmente vengono utilizzati i diagrammi a blocchi o descrizioni del programma in pseudo-linguaggio. Noi ci soffermeremo su questo secondo metodo molto più semplice ed efficace dato che utilizza il linguaggio naturale.

1. Introduci valore variabile A
2. Introduci valore variabile B
3. Se A >= B allora
    3.1. C = A - B
    3.2. C = C / 2
4. Altrimenti
    4.1. C = B - A
    4.2. C = C / 2
5. Stampa C

Esempio di programma in pseudo-linguaggio

Introduzione al linguaggio C

Il C è il risultato di un processo di sviluppo iniziato con un linguaggio chiamato BCPL. Il BCPL ha portato alla nascita del linguaggio B che ha condotto a sua volta allo sviluppo del C negli **anni settanta**. Per molti anni, la versione "standard" del C è stata quella fornita insieme alla versione 5 del sistema operativo UNIX, descritta nel libro *The C Programming Language* di Brian Kernigham e Dennis Ritchie. In seguito alla crescente popolarità dei microcomputer, si sono avute versioni successive del C. Così nacquero differenti versioni del linguaggio e si cercò di creare uno standard, creando una commissione all'inizio dell'estate **1983** con lo scopo di fornire uno standard ANSI del linguaggio C. È proprio l'**ANSI C** che ci accingiamo a studiare.

Cosa è il linguaggio C

Il C viene indicato generalmente come un **linguaggio di medio livello**. Il termine "medio" non indica caratteristiche negative, ma il fatto che il C si pone come via di mezzo fra linguaggi di alto e basso livello: esso unisce elementi dei linguaggi ad alto livello con le funzionalità dell'*assembler*.
Come linguaggio di medio livello, il C consente la gestione di **bit**, **byte** e **indirizzi**. Per tale motivo tale linguaggio è particolarmente adatto alla **programmazione di sistema**: realizzazione di parti di sistemi operativi, assemblatori, interpreti, compilatori, editori e gestori di basi di dati.
Il C è un **linguaggio per programmatori**. Vi sono linguaggi, come il COBOL e il BASIC, nati per fornire ad utenti non programmatori la possibilità di leggere e capire facilmente il significato dei programmi. Il BASIC d'altro canto è nato per dare la possibilità a non programmatori di programmare un computer in maniera da consentire la soluzione di semplici problemi. Il C invece è un vero e proprio linguaggio per programmatori, realizzato da programmatori. Da ai programmatori *poche restrizioni*, *pochi problemi*, *codice veloce* ed *efficienza*. L'**efficienza** dei programmi scritti in C è paragonabile a quella dei programmi scritti in assembler, potendo però usufruire dei vantaggi di un linguaggio ad alto livello.
Una caratteristica fondamentale del C è la sua **portabilità**, ossia la possibilità di adattare il software scritto per un certo tipo di computer a un tipo differente.
Tutti i linguaggi di programmazione ad alto livello introducono il concetto di **tipo di dato**. Un tipo di dato definisce un insieme di valori che una variabile può assumere, associati a un insieme di operazioni che è possibile effettuare su quella variabile. Tipi di dati comuni sono gli interi, i reali e i caratteri. Sebbene il C definisca 5 tipi di dati primari, tra cui i tre citati, **non è un linguaggio fortemente tipizzato**, come il Pascal: infatti esso consente quasi tutte le conversioni fra i tipi, potendo mescolare nelle espressioni tipi differenti. I compilatori C, in generale, eseguono un numero limitato di verifiche di errori, lasciando alla responsabilità del programmatore tale controllo.
Un'altra importante caratteristica del C è il fatto che possiede solo **32 parole chiave** (standard ANSI). Si pensi che il Basic per il PC IBM ne ha ben 159.
Il C è un **linguaggio strutturato**. Consente di definire in modo preciso regole di visibilità dei dati e del codice necessarie all'esecuzione di un compito specifico rispetto al resto del programma. Un modo per definire queste regole e quindi creare moduli di programma è quello di utilizzare sottoprogrammi autosufficienti, ossia sottoprogrammi che definiscono dei dati di cui sono i soli proprietari. L'uso di sottoprogrammi consente di scrivere il codice del sottoprogramma stesso in modo che gli eventi che occorrono al loro interno non provochino alcun *effetto collaterale* su altre parti del programma.
I linguaggi moderni tendono ad essere strutturati e ciò al fine di renderne più semplice la scrittura e manutenzione. Il **mattone** con cui si costruiscono i programmi in C è la **funzione**. Tramite le

funzioni è possibile scomporre il programma in blocchi più piccoli, ottenendo dei **programmi modulari**. Una volta creata e testata una funzione, la si potrà utilizzare in contesti differenti da quello in cui è stata creata con la certezza che fornendole i dati necessari effettuerà il suo compito senza effetti indesiderati sul resto del programma. La creazione di funzioni e quindi di moduli è fondamentale nella realizzazione di programmi di grandi dimensioni con magari un numero elevato di programmatori.
Un'ulteriore possibilità di modularizzazione del codice viene offerta dal C per mezzo dei **blocchi di codice**, che sono blocchi di istruzione logicamente connesse che vengono trattate come un'unità. In C, il blocco di codice si ottiene racchiudendo delle istruzioni tra parentesi graffe aperte e chiuse:

```
if (a>5)
{
        printf("a è maggiore di 5, reinserisci il dato);
        scanf("%d",&a);
}
```

si noti che printf e scanf devono essere eseguite entrambe o non essere eseguite del tutto. I blocchi di codice consentono una **maggiore leggibilità**, **eleganza** ed **efficienza**.