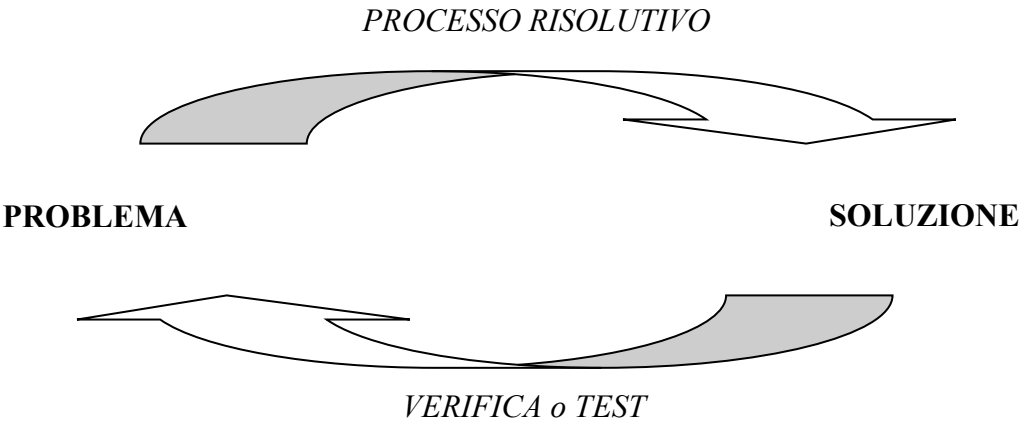
1. DAL PROBLEMA ALL'ALGORITMO

DEF: Un **problema** è un quesito (ossia una domanda) che attende una risposta detta **soluzione** del problema.

Per **processo risolutivo** si intende un insieme di passi da compiere per giungere alla soluzione del problema. Naturalmente deve esistere un modo di controllare l'esattezza della soluzione trovata al problema alla fine del processo risolutivo, ossia deve essere possibile effettuare la **verifica (il test)** della soluzione.

Schematicamente:



Per potere tentare di trovare una soluzione ad u problema quest'ultimo deve essere **correttamente formulato** ossia deve accadere che:

a) non deve apparire evidente che il problema non abbia soluzione;
b) deve esistere un criterio per la verifica del corretto raggiungimento degli obiettivi finali;
c) i dati iniziali devono essere completi.

Un problema è **non risolvibile** quando è un problema correttamente formulato che non ammette soluzioni.

Le figure coinvolte nel processo risolutivo sono due: il **risolutore** e **l'esecutore.**

IL RISOLUTORE è colui che definisce il processo risolutivo per risolvere il problema *Al risolutore spetta l'attività creativa, il processo risolutivo non dipende solo da capacità tecniche ma anche dalla cultura e dall'intuizione del risolutore.*

L'ESECUTORE è colui il quale esegue il processo risolutivo. L'esecutore può essere anche una persona. *All'esecutore spetta l'attività esecutiva del procedimento, ossia seguire il processo risolutivo descritto dal risolutore per giungere concretamente alla soluzione del problema*

Per giungere alla **risoluzione di un problema** occorre effettuarne una sua corretta **formalizzazione** ossia:

- individuare tutte le informazioni di partenza ossia i dati iniziali;
- individuare tutte le informazioni di arrivo ossia lo scopo da raggiungere, la soluzione del problema (i dati finali);
- individuare il processo risolutivo per giungere dalle informazioni di partenza a quelle di arrivo;
- verificare la soluzione trovata.

L'ALGORITMO

Compito dell'algoritmo è quello di formalizzare una serie di "passi" da percorrere o una serie di azioni da svolgere per ottenere i risultati voluti e risolvere un problema seguendo un processo risolutivo.
Gli algoritmi in quanto tali, devono condurre alla risoluzione non del singolo problema, ma di CLASSI DI PROBLEMI.
Risolvere un'intera classe di problemi significa poter mantenere lo stesso insieme di istruzioni guida e poter scegliere invece, fra una diversità di dati iniziali su cui operare.

La definizione di algoritmo

Una prima definizione di algoritmo è stata data dal matematico Markow:
" dicesi *algoritmo* un insieme di istruzioni precise che definisce un procedimento di calcolo con lo scopo di ottenere un determinato risultato".

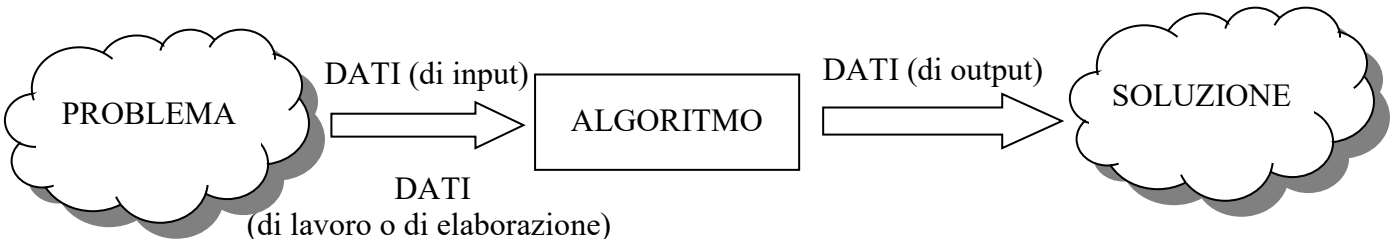
Più precisamente:

DEF: "il termine *algoritmo* contiene la descrizione di un *processo risolutivo* costituita dalla sequenza finita, ordinata ed univocamente interpretabile di *azioni elementari* da eseguire (dette *istruzioni* o *passi* dell'algoritmo) per risolvere una determinata classe di problemi".

Il processo risolutivo descritto da un algoritmo è, come deriva immediatamente da questa definizione, *un processo di tipo sequenziale*.

Se esiste un algoritmo, deve necessariamente esistere un RISOLUTORE e un ESECUTORE.

Nella risoluzione di problemi con l'ausilio del computer il compito del *risolutore* viene svolto dall'**uomo** e quello di *esecutore* dal **computer**



Da subito, rispetto a questa definizione, possiamo precisare che:

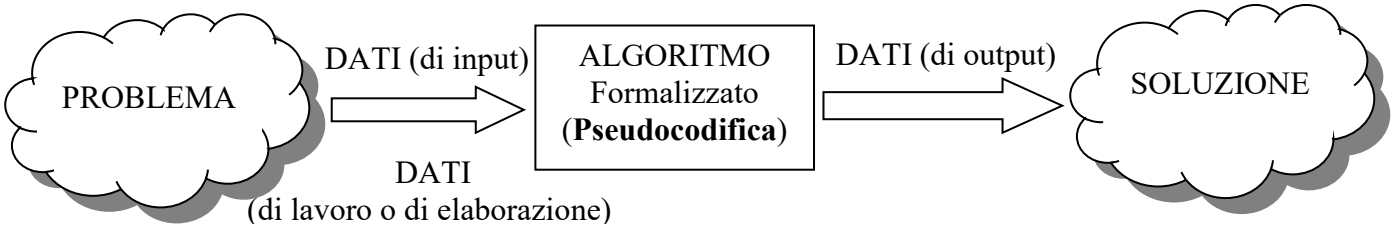
GLI ALGORITMI NON APPARTENGONO SOLO ALLA MATEMATICA
(nella vita quotidiana tutti noi eseguiamo operazioni ben note, che devono essere fatte seguendo una precisa sequenza di istruzioni, alla quale possiamo dare la valenza di algoritmo).

NON BASTA UN ELENCO DI ISTRUZIONI PER AVERE UN ALGORITMO

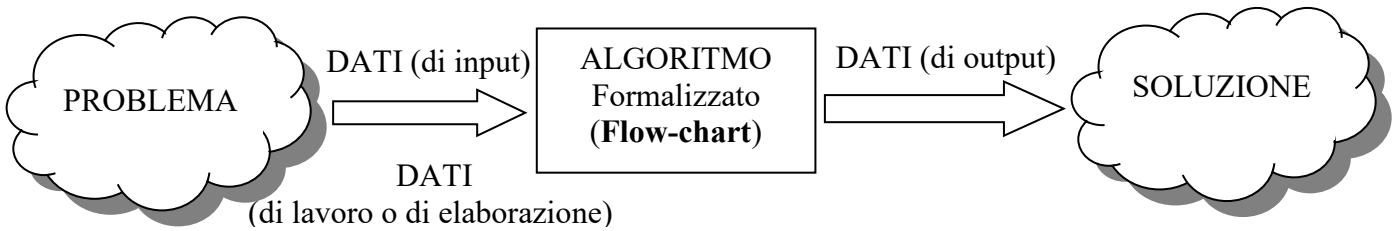
L'algoritmo deve infatti avere le seguenti caratteristiche: essere……..

NON AMBIGUO ossia UNIVOCO	Ogni passo o azione elementare deve essere univocamente interpretabile dall'esecutore
FINITO	Il processo risolutivo descritto dall'algoritmo deve cioè terminare dopo un numero finito di passi o azioni elementari e deve essere caratterizzato da un punto iniziale e da uno finale.
GENERALE	L'algoritmo va inteso come metodo di risoluzione non di un unico caso ma di una classe di problemi di uno stesso tipo
DETTAGLIATO	Ad ogni passo dell'algoritmo deve corrispondere un'azione elementare che l'esecutore è in grado di compiere
DETERMINISTICO	deve risolvere il problema in tutti i suoi aspetti e sempre in tutti i casi. L'algoritmo partendo dalle stesse condizioni iniziali deve fornire sempre gli stessi risultati finali.
COMPLETO	L'algoritmo deve considerare tutti i casi possibili che si possono verificare durante l'esecuzione e per ogni caso indicare la soluzione da seguire.
OSSERVABILE NEI RISULTATI	Deve esserci un riscontro oggettivo dei risultati prodotti dall'algoritmo ossia la verifica deve essere testabile
LIMITATO NEL TEMPO	Anche se impiega molto tempo deve comunque avere un punto di arresto
EFFICIENTE	L'esecuzione dell'algoritmo deve comportare l'utilizzo, tendenzialmente, di meno risorse possibili

Per descrivere un algoritmo bisogna necessariamente utilizzare un linguaggio. La cosa più ovvia sarebbe quella di utilizzare il nostro linguaggio naturale ossia l'italiano ma a causa delle sue possibili ambiguità (che vedremo in seguito) occorre utilizzare i cosiddetti **linguaggi formali**.
Nel nostro caso useremo come linguaggio formale un **linguaggio di progetto** o **pseudolinguaggio** simile all'italiano ma con qualche accorgimento per eliminare le possibili ambiguità:

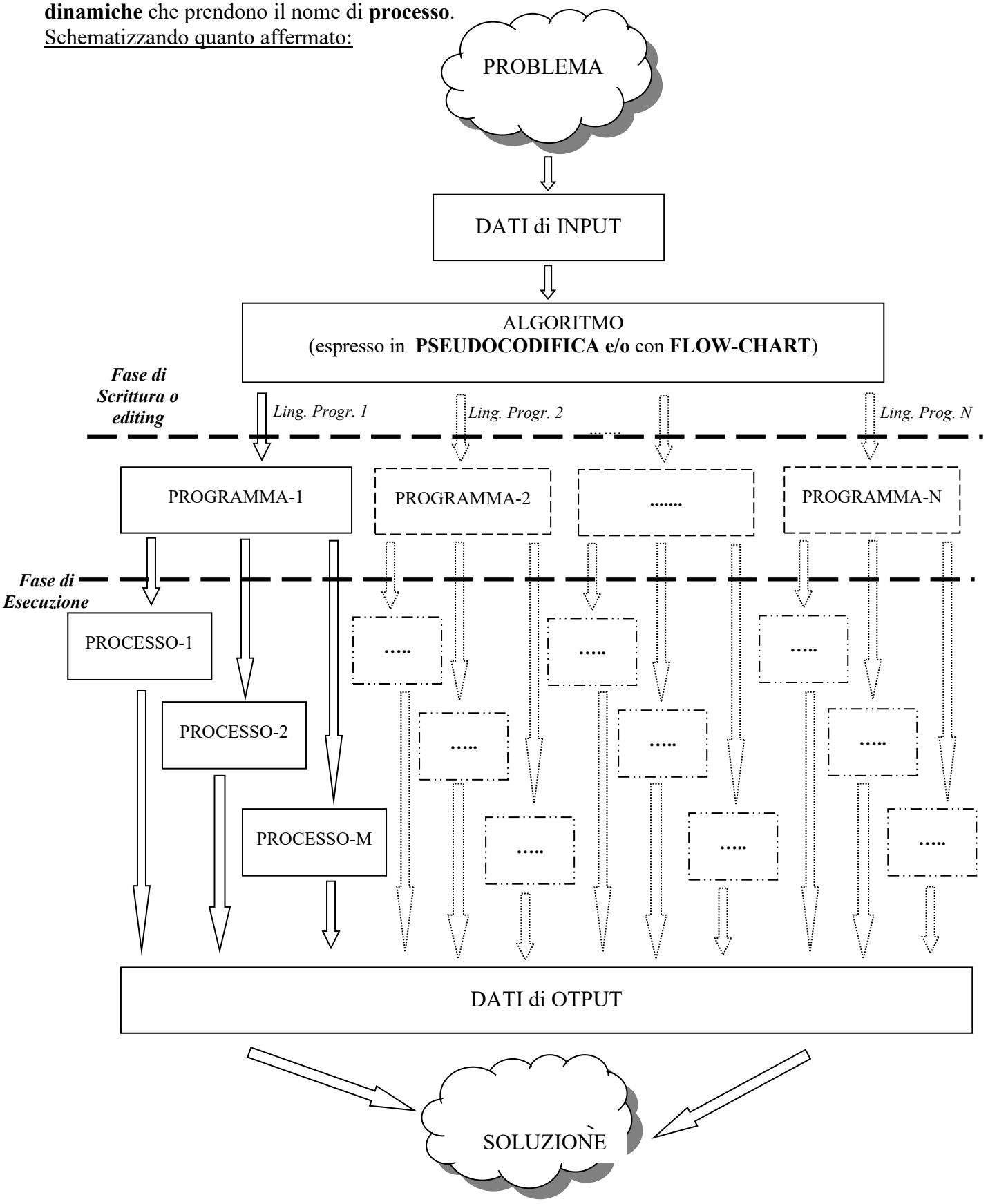


Un secondo linguaggio formale per la rappresentazione dell'algoritmo consiste nell'utilizzare particolari **simboli grafici** (ossia disegni), connessi fra loro, che rappresentano le diverse azioni elementari descritte nell'algoritmo. Questa rappresentazione grafica pende il nome di **"Flow-Chart" o "Diagramma di Flusso":**

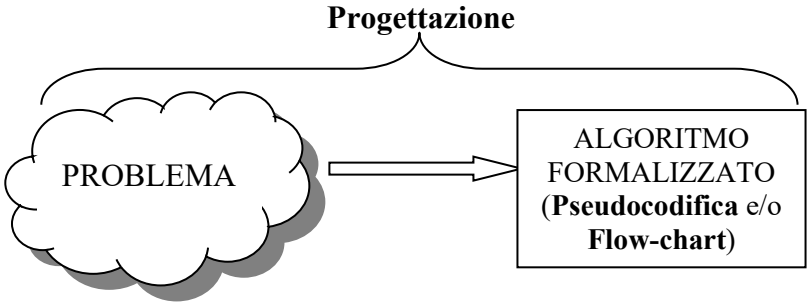


Nel caso in cui **l'algoritmo formalizzato** (espresso in pseudolinguaggio e/o con diagramma di flusso) debba essere eseguito in modo automatico dal computer, vi è la necessità di tradurlo in un linguaggio che possa essere capito ed eseguito dal computer cioè in **linguaggio binario o linguaggio macchina** (costituito dai soli simboli 0 ed 1).
Tale attività veniva fatta una volta direttamente dall'uomo (e per alcune attività tecniche può ancora esser fatta) oppure può essere effettuata da particolari programmi (**compilatori ed interpreti**) che hanno il compito di tradurre in linguaggio macchina un algoritmo formalizzato scritto in un

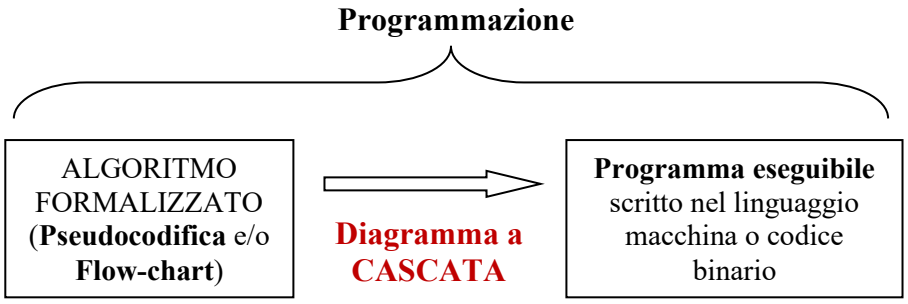
linguaggio di programmazione (che prende il nome di **programma)**. L'attività di traduzione prende il nome di **codifica** o **implementazione** dell'algoritmo.
Un programma è una **entità statica**, mentre la sua esecuzione richiedendo la presenza di un esecutore, di dati iniziali e di determinate risorse macchina rappresenta **una o più entità dinamiche** che prendono il nome di **processo**.
Schematizzando quanto affermato:



Per risolvere un problema attraverso l'uso di un computer abbiamo visto che la prima fase è quella di passare dal **problema** all'**algoritmo formalizzato** (ossia scritto usando la pseudocodifica o utilizzando il flow-chart che sono **linguaggi formali**).
Questa fase prende il nome di **fase di progettazione** o più brevemente **progettazione** dove si esaminano le possibili soluzioni del problema e dopo averle confrontate si sceglie quella considerata migliore.



La fase successiva consiste nel tradurre l'algoritmo formalizzato in una serie di istruzioni scritte in un linguaggio che la macchina (ossia il nostro esecutore) sia in grado di comprendere.
Questa fase prende il nome di **fase di programmazione** o più brevemente **programmazione** e può essere riassunta come segue:

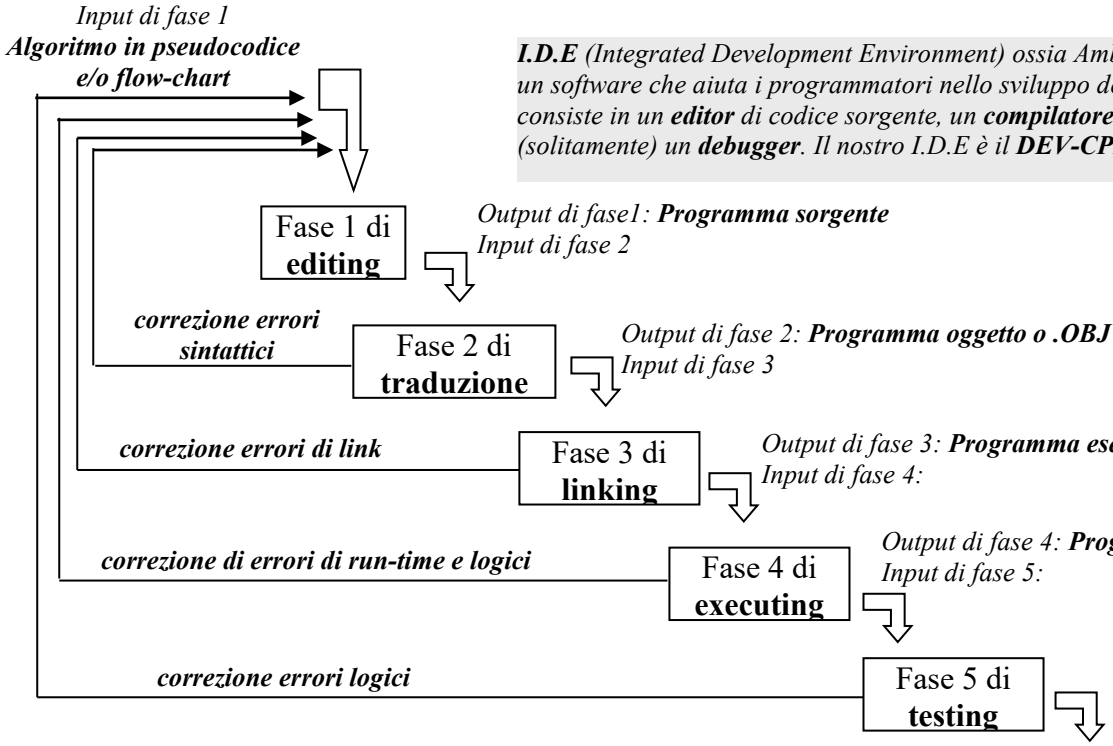


I **linguaggi di programmazione di alto livello** (esempio C, C++, Visual Basic, Java, etc.) sono linguaggi più vicini al linguaggio naturale le cui istruzioni sono indipendenti dalla macchina e permettono di descrivere l'idea che sta dietro all'operazione.
Quindi essi permettono di scrivere il **programma sorgente** che è la traduzione di un algoritmo formalizzato in un ben determinato linguaggio di programmazione di alto livello.
Ma il programma sorgente non è scritto ancora in un linguaggio comprensibile da un elaboratore.
Un elaboratore riconosce solo istruzioni scritte in codice binario (ossia utilizzando sequenze di 0 ed 1), il cosiddetto **linguaggio macchina,** ossia un linguaggio a basso livello dipendente dal tipo di macchina utilizzato.
Una volta i programmatori immettevano direttamente negli elaboratori le istruzioni in linguaggio macchina. L'operazione era ovviamente laboriosa e complicata, per cui nel tempo si sono sviluppati e diffusi i linguaggio di programmazione (ad alto livello) sempre più vicini al modo di ragionare del'uomo affiancandogli l'utilizzo di particolari programmi "traduttori" (**compilatori** ed **interpreti**) che svolgessero il difficile compito di tradurre ogni istruzione di alto livello in una sequenza binaria.

Compilatori ed **interpreti** sono quindi entrambi programmi in grado di *tradurre* le istruzioni di un programma scritto in un linguaggio di programmazione ad alto livello detto **programma sorgente,** in istruzioni comprensibili al computer ossia in istruzioni di un programma scritto in **linguaggio macchina** detto **programma oggetto** (sequenze di '0' ed '1' comprensibili per l'elaboratore).
Sono in pratica dei programmi **traduttori** ma sono profondamente diversi nel funzionamento perché:

- I **compilatori** sono programmi che accettano in input un intero programma scritto in linguaggio ad alto livello (*programma sorgente*) e lo traducono interamente in un programma scritto in linguaggio macchina (*programma oggetto*);
- Gli **interpreti** sono programmi che accettano in input le singole istruzioni di un programma scritto in linguaggio ad alto livello e le traducono una alla volta in singole istruzioni in linguaggio macchina.

DIAGRAMMA a CASCATA: Dettaglio delle fasi di programmazione



***I.D.E** (Integrated Development Environment) ossia Ambiente di Sviluppo Integrato è un software che aiuta i programmatori nello sviluppo del codice. Normalmente consiste in un **editor** di codice sorgente, un **compilatore** e/o un **interprete**, un **linker**, e (solitamente) un **debugger**. Il nostro I.D.E è il **DEV-CPP**.*

***Programma di scrittura o "editor"** è il programma che permette di scrivere il programma sorgente eventualmente guidando, con l'aiuto di colori, font, etc,., al rispetto della sintassi del linguaggio di programmazione utilizzato.*

***Programma compilatore o "compiler"** è il programma che traduce in una volta sola TUTTO il programma sorgente compiendo l'analisi lessicale, sintattica e semantica.*

***Programma interprete o "interpreter"** è il programma che traduce una istruzione alla volta il programma sorgente compiendo l'analisi lessicale, sintattica e semantica.*

***Programma eseguibile o .EXE** è il programma oggetto collegato con i moduli oggetto delle funzioni di libreria e/o utente utilizzate e previste nel linguaggio di programmazione di alto livello scelto.*

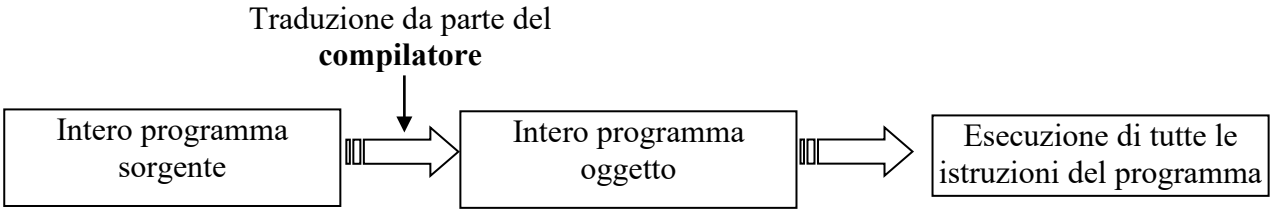
*Output di fase 5: **Programma eseguibile o EXE definitivo corretto***

***Programma di link o 'linker'** è il programma che collega il programma oggetto con i moduli oggetto delle funzioni di libreria e/o utente utilizzate e previste nel linguaggio di programmazione utilizzato.*

***Funzione di libreria** è un programma oggetto già compilato fornito dal linguaggio di programmazione scelto predisposto per essere collegato al programma oggetto che si sta creando. Di solito permette di risolvere problemi di programmazione comuni (Es. libreria matematica, libreria per la gestione dell'I/O,libreria per la gestione delle stringhe, etc.)*

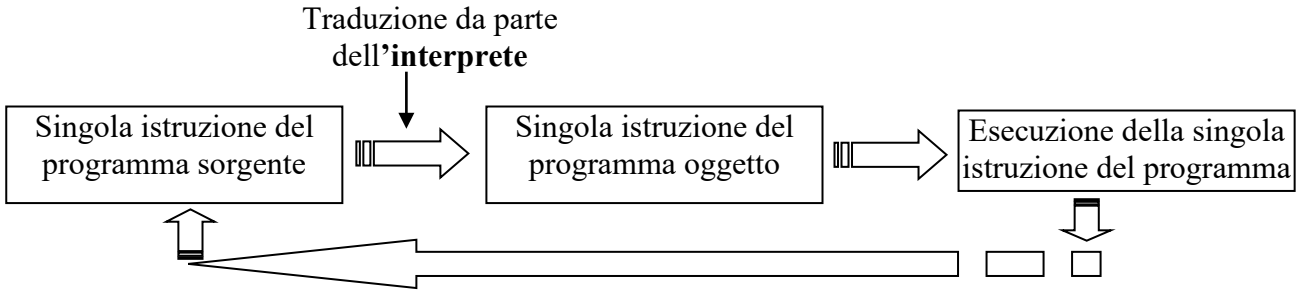
***Programma di debugging o 'debugger** è un programma specificatamente progettato per l'analisi e l'eliminazione dei bug (ovvero errori di programmazione interni al programma) presenti in altri programmi.*

Schema della fase di **compilazione** di un programma sorgente



Il compilatore traduce tutte le istruzioni del programma una dopo l'altra ma, se c'è qualche errore di sintassi, non viene generato il programma eseguibile (non viene eseguita nessuna istruzione).

Schema della fase di **interpretazione** di un programma sorgente



L'interprete traduce la prima istruzione del programma e, se non c'è alcun errore di sintassi, la esegue;
L'interprete traduce la seconda istruzione del programma e, se non c'è alcun errore di sintassi, la esegue;
……
L'interprete traduce l'ultima istruzione del programma e, se non c'è alcun errore di sintassi, la esegue.

Compilatori ed interpreti:vantaggi e svantaggi

	COMPILATORE	**INTERPRETE**
vantaggi	Il programma sorgente viene tradotto una volta sola	Il programma sorgente può essere modificato interattivamente riducendo il debugging
	L'esecuzione del programma è più veloce in quanto la fase di traduzione è già avvenuta	
	Il programma compilatore non deve risiedere in memoria centrale	
	E' possibile garantire la segretezza del programma sorgente (delle linee di codice)	
svantaggi	Ogni volta che si modifica il programma sorgente occorre ricompilare	Il programma sorgente viene tradotto ogni volta che deve essere eseguito
		L'esecuzione del programma è più lenta in quanto avviene assieme alla traduzione
		Il programma interprete deve risiedere in memoria centrale
		Non è possibile garantire la segretezza del programma sorgente

ALGORITMO = DATI + ISTRUZIONI

Un algoritmo (e quindi dopo le trasformazini necessarie un programma) può essere in sintesi visto come un manipolatore di dati.

A fronte di dati in input che caratterizzano il problema del quale ne descrive il processo risolutivo, produce dati di output come risultato del problema stesso.

Le operazioni sui dati si chiamano "istruzioni".
I "dati" sono gli oggetti su cui vengono eseguite le istruzioni.

DATI

Innanzitutto specifichiamo la differenza tra i termini **dato** ed **informazione**.

Il termine **dato** deriva dal termine latino *datum* (plurale *data*) ossia *fatto*. Lo scopo dei dati è quello di rappresentare in vari modi i fatti ritenuti importanti nell'ambito di una certa realtà di interesse.
Una **informazione** invece è la variazione di conoscenza che può essere acquisita (o *ottenuta*) dai dati.

Da questa considerazione segue che i dati non possono essere ritenuti utili (ossia non danno informazione) fino a quando non si fornisce una **chiave di interpretazione** (o semplicemente interpretazione) che permetta di comprendere il loro **significato** (o semantica) ossia i fatti che essi codificano.

In informatica inoltre i **dati** sono caratterizzati da un **nome** e da un **tipo.**

Il **nome o identificatore** serve per identificare univocamente il dato. E' buona norma utilizzare nomi che consentano di richiamarne immediatamente il significato.

Il **tipo** è legato al contenuto e quindi ai **valori** che il dato può assumere ed alle operazioni che possono essere eseguite su di **essi**.

CLASSIFICAZIONE DEI DATI secondo tre punti di vista

1) A seconda di come interagiscono con il computer (e quindi con l'algoritmo e di conseguenza con il programma) i dati si classificano in:

- **dati di input**: sono quei dati forniti dall'esterno e che servono per la risoluzione del problema;
- **dati di output**:sono quelli che vengono comunicati all'esterno come soluzione del problema;
- **dati di lavoro (o di elaborazione)**: sono quelli utilizzati durante l'esecuzione del processo risolutivo (in aggiunta ai dati di input);

2) In funzione degli oggetti che rappresentano (quindi del loro **tipo**) i dati si classificano in:

- **dati numerici**: sono quei dati che contengono numeri e che possono essere utilizzati in formule matematiche. Tra di questi è possibile ancora distinguere:
  (a) dati **numerici interi:** sono quei dati che prevedono numeri senza cifre decimali (che in pseudocodifica indicheremo con **INT**);
  (b) dati **numerici reali:** sono quei dati che prevedono numeri con cifre decimali (che in pseudocodifica indicheremo con **REAL**);

- **dati alfanumerici**: sono quei dati che contengono sia caratteri (cifre, lettere, caratteri speciali, simboli di interpunzione, etc.) che singole cifre, ma sulle quali non è possibile effettuare alcun calcolo matematico. Tra di questi è possibile ancora distinguere:

(a) dati **alfanumerici** formati da **un singolo carattere:** sono quei dati che prevedono come valore un qualunque carattere singolo (secondo lo standard ASCII) racchiuso tra apice singolo (che in pseudocodifica indicheremo con **CHAR**);
(b) dati **alfanumerici** formati da un **insieme di caratteri** (**stringhe**): sono quei dati che prevedono un insieme di caratteri (secondo lo standard ASCII) racchiusi tra doppi apici (che in pseudocodifica indicheremo come **ARRAY DI CHAR**);

- **dati logici o booleani** sono quei dati che contengono i possibili valori di verità (**VERO** o **FALSO** da indicare senza l'uso dei doppi apici) (che in pseudocodifica indicheremo con **BOOL**);

3) In funzione della possibilità di cambiamento del loro valore i dati si classificano in:

- **dati variabili o variabili**: sono quei dati che possono cambiare il loro valore durante il processo risolutivo;
- **dati costanti o costanti**: sono quei dati che non possono cambiare il loro valore durante il processo risolutivo.

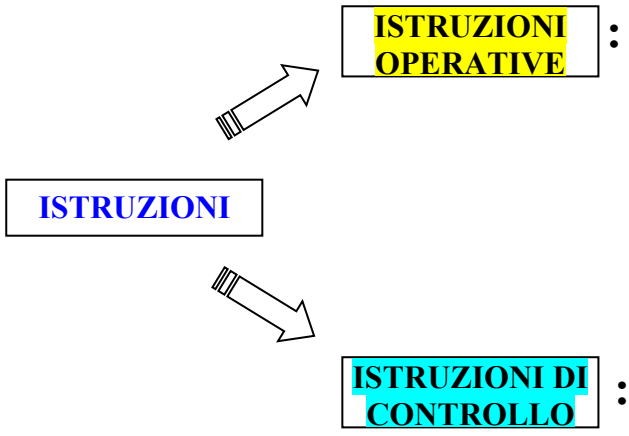
Schematizzando possiamo legare il concetto di **variabile** a quello di **contenitore aperto** all'interno del quale è possibile inserire dei valori e sostituirli con degli altri purchè dello stesso tipo.
Invece possiamo legare il concetto di **costante** a quello di **contenitore chiuso** all'interno del quale non sarà possibile modificare il dato presente, ma se ne può sempre usare il contenuto

N.B. Il concetto di **contenitore** va legato al concetto di **zona di memoria** adibita a contenere temporaneamente ossia per tutto il corso di elaborazione determinati valori.

LE ISTRUZIONI

Abbiamo detto che ogni algoritmo contiene la descrizione di un processo risolutivo formata da un insieme finito di istruzioni elementari (o azioni o passi) da dare all'esecutore per trasformare i dati iniziali o di **input** nei dati finali o di **output**.

Tra le istruzioni distinguiamo:



Sono le istruzioni che corrispondono ad azioni direttamente eseguibili dall'elaboratore
Esse sono:

1. **Dichiarazione di costanti e variabili;**
2. **Istruzioni di input/output;**
3. **Istruzione di assegnazione**

e possono riguardare:
la capacità di:

- **memorizzazione,**
- **di comunicazione**
- **di elaborazione**

Sono le istruzioni che permettono di controllare il flusso delle istruzioni di un algoritmo eventualmente instradando percorsi differenti durante l'esecuzione, in funzione del verificarsi o meno di determinate condizioni. Esse sono

1. **istruzione (o costrutto) di SEQUENZA;**
2. **istruzione (o costrutto) di SELEZIONE;**
3. **istruzione (o costrutto) di ITERAZIONE.**

Istruzioni OPERATIVE

Sono le istruzioni che corrispondono ad azioni direttamente eseguibili dall'elaboratore (in realtà dal microprocessore o CPU) e che possono riguardare:

1. **la capacità di memorizzazione**
(ossia la capacità di riservare uno spazio di memoria di lunghezza opportuna per una variabile e/o una costante)
2. **la capacità di comunicazione**
(ossia l'acquisizione dei dati in ingresso e la visualizzzione dei dati in uscita)
3. **la capacità di elaborazione**
(ossia la capacità di sapere effettuare la valutazione o il calcolo di un'espressione oppure sapere effettuare la codifica/decodifica di un dato a seconda del tipo posseduto)

Dichiarazione di una costante

Viene implicitamente dichiarata scrivendo nell'ambiente globale di un algoritmo:

<nome_costante> <valore_costante>

Essa sarà dettagliata all'interno della tabella dei dati di lavoro (o elaborazione).

Es. **PIGRECO 3.14** dichiara (per poi poterla utilizzara) una costante REALE con valore pari a 3.1

Flow-chart istruzione di dichiarazione delle costanti

N.B. Le dichiarazioni implicite delle costanti non vengono rappresentate in un flow-chart

Dichiarazione di una variabile

L'istruzione di dichiarazione di una variabile, presente in quasi tutti i linguaggi di programmazione, consente di definire all'interno di un algoritmo un dato di input, di output o di lavoro che sia variabile.
Non si applica nel caso di costanti che non devono essere dichiarate esplicitamente all'interno dell'algoritmo, ma che verranno implicitamente definite una volta dettagliate all'interno della tabella dei dati di lavoro (o elaborazione).

La sua pseudocodifica è

<variabile1> [,<variabile2>,…<variabilen>] : <tipo>

Es. **a, b : INT** dichiara (per poi poterle utilizzare) due variabili di nome rispettivamente **a** e **b** entrambe di tipo **intero** (ossia potranno assumere solo valori interi e sarà possibile effettuare su di loro solo le operazioni previste per i numeri interi)

Flow-chart istruzione di dichiarazione delle variabili

N.B. Le istruzioni di dichiarazione delle variabili non vengono rappresentate in un flow-chart

Istruzione di input/output

L'istruzione di input, presente in tutti i linguaggi di programmazione, consente di attribuire dall'esterno un valore ad una variabile tra quelli possibili per il suo tipo.

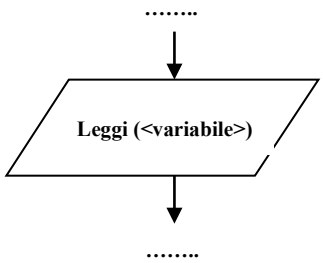
La sua pseudocodifica è

Leggi (<variabile>)

Es. **Leggi (a)**
Significato: acquisisci dall'esterno (tramite tastiera) un valore ammesso per la variabile il cui nome o identificatore è **a**.

Flow-chart istruzione di input

Nel flow-chart è rappresentata da un **parallelogramma** al cui interno è scritta la pseudocodifica dell'istruzione



L'istruzione di output, presente in tutti i linguaggi di programmazione, consente di comunicare all'esterno il **valore** ad una **variabile** o di una **costante** tra quelli possibili per il suo tipo.

La sua pseudocodifica è

Scrivi (<variabile>|>costante>)

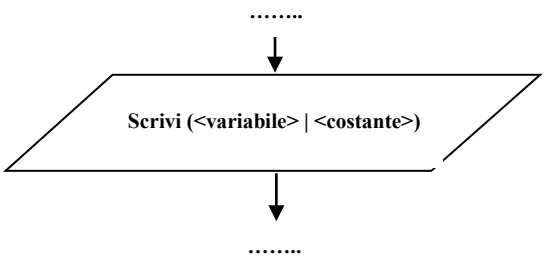
Es. **Scrivi (a)**
Significato: fornisci dall'esterno (tramite video) il valore calcolato per la variabile il cui nome o identificatore è **a**.

Es. **Scrivi (PIGRECO)**
Significato: fornisci dall'esterno (tramite video) il valore calcolato per la costante il cui nome o identificatore è **PIGRECO**

Es. **Scrivi ("Inserisci valore della variabile:")**
Significato: fornisci dall'esterno (tramite video) il valore della stringa costante **"Inserisci valore della variabile"**

Flow-chart istruzione di output

Nel flow-chart è rappresentata da un **parallelogramma** al cui interno è scritta la pseudocodifica dell'istruzione



Istruzione di assegnazione

L'istruzione di assegnazione presente in tutti i linguaggi di programmazione consente (come anche l'istruzione *Leggi()*) di attribuire un valore ad una variabile.

Essa è in pratica un operatore di tipo binario identificato dal simbolo ← che viene detto **operatore di assegnazione**.

In generale quando viene scritto:

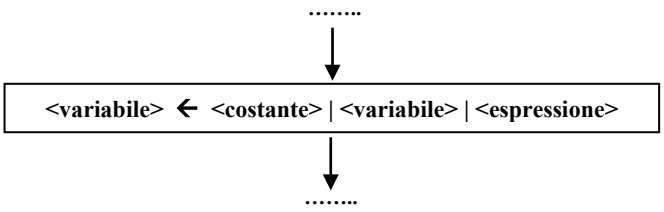
<variabile> ← <costante> | <variabile> | <espressione>

il primo termine a sinistra contiene sempre una variabile mentre quello a destra può essere una costante, un'altra variabile oppure *un'espressione*.

DEF: In generale, per espressione si intente un insieme di costanti e variabili sulle quali si effettuano una serie di operazioni possibili in accordo al tipo di dati posseduto

Flow-chart istruzione di "Assegnazione"

Nel flow-chart è rappresentata da un **rettangolo** al cui interno è scritta la pseudocodifica dell'istruzione



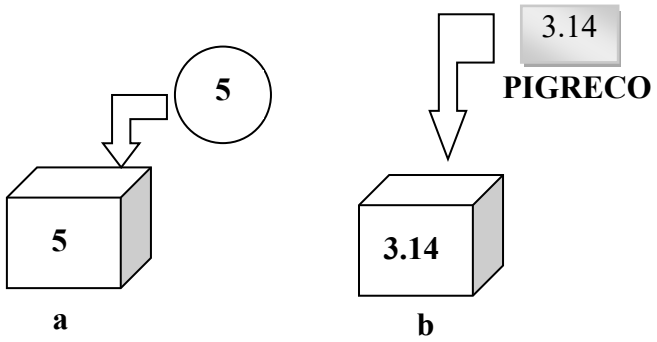
Esempi possibili di assegnazione:

1) <variabile> ← <costante>

Es. **a ← 5**

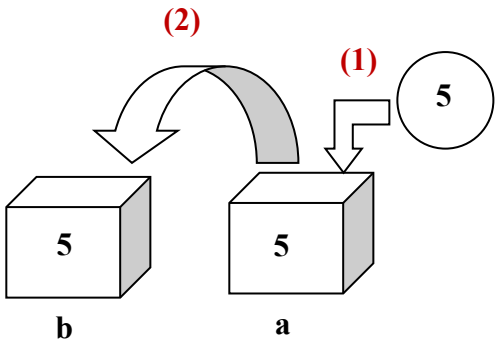
oppure

b ← PIGRECO



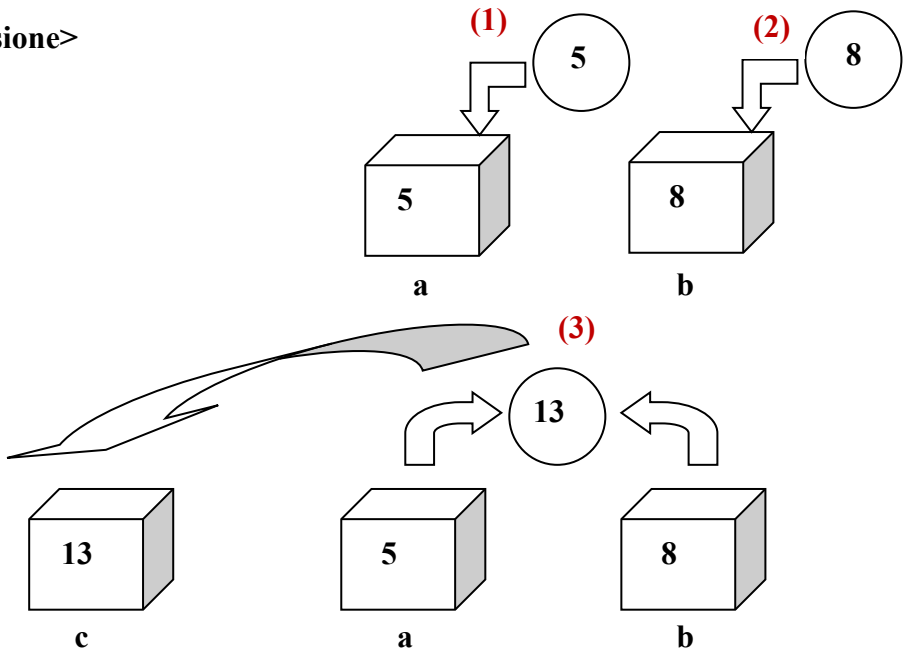
2) <variabile> ← <variabile>

Es. **a ← 5** **(1)**
b ←a **(2)**



3) <variabile> ← <espressione>

Es. **a ← 5** **(1)**
b ← 8 **(2)**
c ← a + b **(3)**



N.B. LA PRIMA COSA DA FARE NELLA PROGETTAZIONE DI UN ALGORITMO, PRIMA DI PROCEDERE CON IL DETTAGLIO DELLE SUE ISTRUZIONI, È DETTAGLIARE LE SEGUENTI TABELLE CHE FORNISCONO L'ELENCO DI TUTTI I DATI (VARIABILI e/o COSTANTI) NECESSARI ALLO SVOLGIMENTO DEL SUO PROCESSO RISOLUTIVO.

TABELLE RIASSUNTIVE PER L'ANALISI DEI DATI

DATI DI INPUT DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile (1)	**Tipo dati (2)**	**Tipo Allocazione (3)**	**Valori ammessi (4)**	**Descrizione (5)**

DATI DI OUTPUT DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

DATI DI ELABORAZIONE (LAVORO) DEL PROBLEMA PRINCIPALE (PROCEDURA MAIN)				
Nome variabile oppure **nome costante (6)**	**Tipo dati**	**Tipo Allocazione**	**Valori ammessi**	**Descrizione**

(1) Il **nome di una variabile** deve essere scritto tutto **con caratteri minuscoli** senza utilizzare spazi e caratteri speciali (quali &, *,#, @).
In caso di nomi di variabili costituiti da più di una parola è possibile utilizzare il carattere '-' (trattino) oppure '_' (underscore).
Esempio: Sono nomi di variabili valide i seguenti: ***a, pippo, somma-2, somma_2***
Sono nomi di variabili non valide i seguenti: ***A, Pippo, somma 2, somma$2***

(2) I **tipi** utilizzabili sono i tipi di **dato semplice** (un solo valore possibile per una variabile in un certo istante durante 'esecuzione dell'algoritmo) o tipo di **dato strutturato o struttura dati** (più valori possibili per una variabile in un certo istante durante l'esecuzione dell'algoritmo)
I tipi di dati **semplice** che utilizzeremo sono:

INT: per i numeri interi relativi (eventualmente con il segno) ma senza parte decimale; i valori possibili per questo tipo di variabile sono i normali interi numeri scritti nella modalità consueta;
Esempio: 12 -45 +32

REAL: per i numeri reali (eventualmente con il segno) con parte decimale ; i valori possibile per questo tipo di variabile sono i normali numeri reali scritti nella modalità consueta per i quali si consiglia l'uso, al posto della virgola decimale, del PUNTO DECIMALE;
Esempio: 12.34 -45.00 +32.1 ma anche 12 -45 +32

CHAR: per il singolo carattere alfanumerico secondo la codifica ASCII estesa: i valori possibili per questo tipo di variabile sono i caratteri racchiusi da due apici singoli;
Esempio: 'a' 'M' '$' '@' ';' 'b' |

BOOL: per il singolo valore di verità dell'algebra booleana: i valori possibili per questo tipo di variabile sono i seguenti: **VERO** oppure **FALSO**;

(3) Da valorizzare con:

STATICA per indicare l'allocazione della variabile di tipo statico (ossia costante per tutta la durata dell'algoritmo)

DINAMICA per indicare l'allocazione della variabile di tipo dinamico (ossia modificabile nel corso dell'esecuzione dell'algoritmo)

N.B. per questa prima fase useremo variabili esclusivamente STATICHE per le quali, quindi, occorre valorizzare con STATICA la colonna "Tipo Allocazione" in attesa di poter utilizzare anche dati allocati dinamicamente.
Le costanti sono da intendersi sempre ad allocazione statica.

(4) In questa colonna occorre indicare, utilizzando se possibile il linguaggio della matematica, le condizioni che i valori di quella variabile devono rispettare nel corso dell'algoritmo.
*Esempio: Se **a** è la variabile **STATICA** di tipo **INT** che rappresenta il primo coefficiente di un'equazione di secondo grado è ovvio che dovrà essere scritta, nella colonna "Valori ammessi", la condizione $a \neq 0$*

(5) Da valorizzare indicando brevemente ma in modo significativo la spiegazione del significato della variabile.

(6) Il **nome di una costante** deve essere scritto **tutta con caratteri MAIUSCOLI** senza utilizzare spazi e caratteri speciali (quali &, *,#, @).
In caso di nomi di costanti costituiti da più di una parola è possibile utilizzare il carattere '-' (trattino) oppure '_' (underscore).
*Esempio: Sono nomi di costanti valide i seguenti: **PIGRECO, PI-GRECO, PI_GRECO***
*Sono nomi di costanti non valide i seguenti: Pigreco **PI+GRECO** oppure **PI&GRECO***

SCHEMA DELLA PSEUDOCODIFICA DI UN ALGORITMO

ALGORITMO <nomealgoritmo>

…….
…….
…….

AMBIENTE GLOBALE: le costanti, le variabili o i tipi di dato dichiarati in questa zona saranno visibili (ossia utilizzabili) non solo dalla procedura main ( ) , ma anche da tutte le altre procedure in cui l'algoritmo sarà stato suddiviso.
N.B. Non è possibile inserire nessun altro tipo di istruzione

PROCEDURA main ( )
……..
……..

AMBIENTE LOCALE: le variabili o i tipi di dato dichiarati dichiarate in questa zona saranno visibili (ossia utilizzabili) SOLO dalla procedura main ( ).
N.B. Non è possibile inserire nessun altro tipo di istruzione

INIZIO

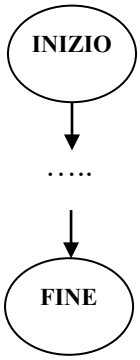
……..
……..
……..…
…….
…….
…….

CORPO DELLA PROCEDURA: contiene tutte le istruzioni (OPERATIVE E DI CONTROLLO) che descrivono il processo risolutivo dell'algoritmo TRANNE le istruzioni di dichiarazione di costanti, variabili e tipi di dato

RITORNA
FINE

Flow-chart struttura ALGORITMO

Il flow-chart di un algoritmo è uno **schema grafico** in cui vengono rappresentate tutte le istruzioni progettate comprese tra la stringa "INIZIO" e la stringa "FINE" (rappresentate con un'ellisse)



ISTRUZIONI (o COSTRUTTI) DI CONTROLLO

a) Istruzione o Costrutto di "**SEQUENZA"**:

La sequenza è il costrutto più semplice tra i tre costrutti fondamentali della programmazione strutturata. Si utilizza quando le azioni ossia le istruzioni devono essere seguite ordinatamente una dopo l'altra senza alcuna possibilità di scelta.

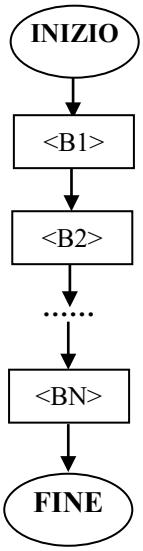
Pseudocodifica "Sequenza":

```
.....
INIZIO
   <B1>
   <B2>
    …..
    <BN>
FINE
.....
```

<B1>, <B2> …<BN> sono **blocchi semplici** o **blocchi composti** di istruzioni.
Un *blocco semplice* di istruzioni è formato da UNA SOLA istruzione operativa o di controllo.
Un *blocco composto* di istruzioni è invece formato da PIU' blocchi semplici.

N.B. Per questo motivo anche un intero algoritmo strutturato (costituito solo attraverso i tre costrutti fondamentali di sequenza, selezione ed iterazione) può essere visto come la sequenza di uno o più blocchi semplici e/o composti.

Flow-chart istruzione di "SEQUENZA"



b) Istruzione o Costrutto di "**SELEZIONE"**:

La selezione ci permette di instradare uno, due o più percorsi (vie o strade o rami) rispetto al flusso dell'algoritmo entrante.

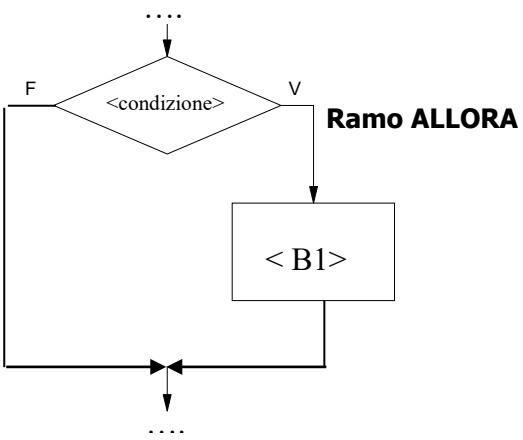
In un primo momento distinguiamo tra le istruzioni di selezione che consentono *effettuare una scelta* fra due possibili alternative: la selezione unaria e la selezione binaria (LOGICA BOOLEANA).
Per effettuare la scelta dobbiamo *valutare una condizione booleana* (che ovviamente può assumere solo due possibili valori: vero o falso)

Pseudocodifica "Selezione UNARIA" o "A UNA VIA":

```
.....
SE <condizione>
   ALLORA
      <B1>
FINE SE
.....
```

<condizione> è un qualunque enunciato (semplice o composto) dell'Algebra di Boole
<B1> è un **blocco semplice** o **blocco composto** di istruzioni
Se la condizione è VERA si esegue il blocco di istruzioni <B1>
Se la condizione è FALSA si prosegue normalmente**.**

Flow-chart istruzione di "Selezione unaria" o "A UNA VIA":

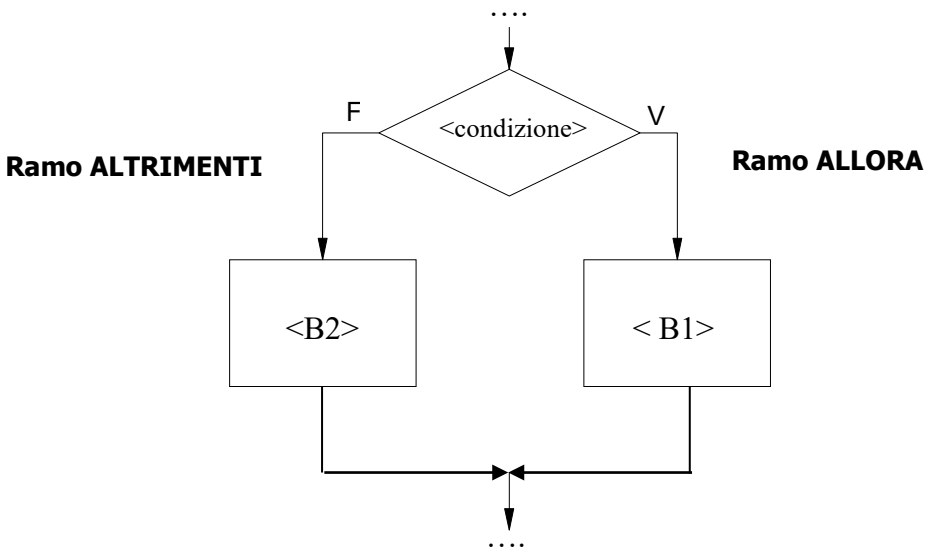


Pseudocodifica "Selezione binaria" o "A DUE VIE":

....
SE <condizione>
ALLORA
<B1>
ALTRIMENTI
<B2>
FINE SE
....

<condizione> è un qualunque enunciato (semplice o composto) dell'Algebra di Boole
<B1> e <B2> sono **blocchi semplici** o **blocchi composti** di istruzioni
Se la condizione è VERA si esegue il blocco di istruzioni <B1>
Se la condizione è FALSA si esegue il blocco di istruzioni <B2>

Flow-chart istruzione di "SELEZIONE BINARIA" o "A DUE VIE":



Esiste anche un'istruzione di selezione che permette *di effettuare una scelta* fra n possibili valori ed instradare di conseguenza n possibili percorsi distinti rispetto al flusso dell'algoritmo entrante.
Per effettuare la scelta dobbiamo *valutare il valore di una variabile o di una espressione*

Pseudocodifica "Selezione multipla o n-ria":

.....
NEL CASO CHE <var> | <espr> **SIA**
<valore1> **:** <B1>
<valore 2> **:** <B2>
…… : ……
<valoreN> **:** <BN>
[ALTRIMENTI : <BN + 1>**]**
FINE CASO
.....

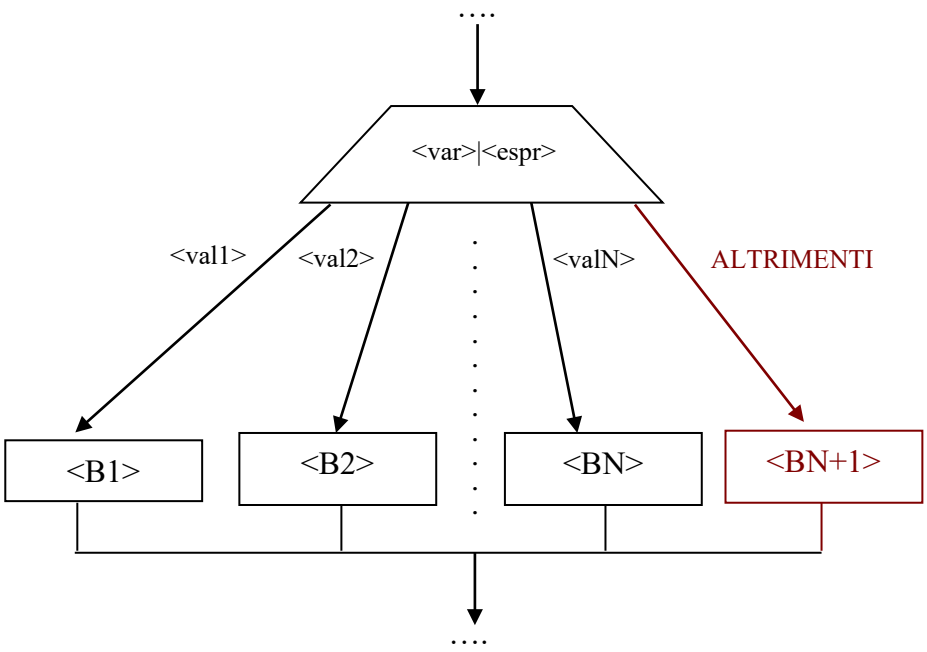
<*var*> | <*espr*> può essere una variabile oppure una espressione che può assumere un unico valore numerico intero ma non può mai essere espresso come risultato di una condizione logica.
<*valorex*> può essere

- una costante di uguale valore a <var> | <espr>;
- un valore unico;
- una lista di valori;

N.B. E' opportuno ricordare che <valorex> devono essere dello stesso tipo di <var> | <espr>

N.B. e' possibile avere anche liste di valori separati da virgole

Flow-chart istruzione di "Selezione multipla o n-aria"



b) Istruzione o Costrutto di "**ITERAZIONE"**:

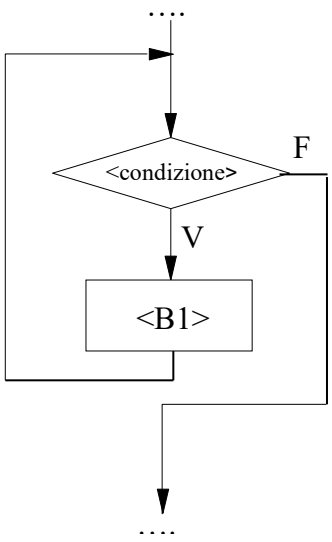
Il **costrutto iterazione o iterativo** viene utilizzato quando una istruzione (o un gruppo di istruzioni.) deve essere eseguita finchè non si verifica una determinata condizione.

Pseudocodifica *istruzione di "Iterazione con controllo in testa (o PRE-CONDIZIONALE)"*::

```
.....
MENTRE <condizione> ESEGUI
      <B1>
FINE MENTRE
.....
```

<condizione> è un qualunque enunciato (semplice o composto) dell'Algebra di Boole.
<B1> è un **blocco semplice** o un **blocco composto** di istruzioni**.**
Se la condizione è VERA si esegue il blocco di istruzioni <B1>
Se la condizione è FALSA si arresta il processo iterativo.
Se la condizione è inizialmente FALSA il ciclo non viene mai eseguito

Flow-chart *istruzione di "Iterazione con controllo in testa (o PRE-CONDIZIONALE)":*

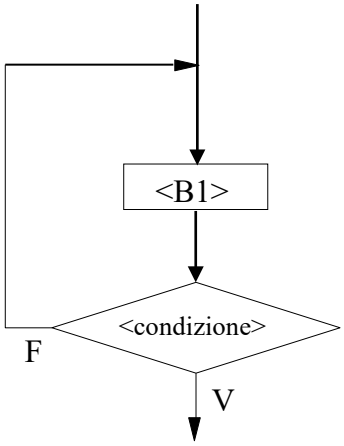


Pseudocodifica *istruzione di "Iterazione con controllo in coda (o POST-CONDIZIONALE)"*:

```
.....
RIPETI
      <B1>
FINCHE' <condizione>
.....
```

<condizione> è un qualunque enunciato (semplice o composto) dell'Algebra di Boole.
<B1> è un **blocco semplice** o un **blocco composto** di istruzioni.
Se la condizione è FALSA si esegue il blocco di istruzioni <B1>.
Se la condizione è VERA si arresta il processo iterativo.
Il blocco <B1> di istruzioni viene eseguito ALMENO una volta perché la condizione viene testata dopo la sua esecuzione.

Flow-chart *istruzione di "Iterazione con controllo in coda (o POST-CONDIZIONALE)":*



Mettiamo a confronto i due costrutti iterativi introdotti finora:

COSTRUTTO ITERAZIONE MENTRE	**COSTRUTTO ITERAZIONE RIPETI …. FINCHE'**
1) La condizione viene controllata **prima** di eseguire il blocco di istruzioni <B1>; 2) Può anche **non eseguire mai** il blocco di istruzioni <B1>; 3) Esegue il blocco di istruzioni <B1> fino a che la condizione in testa risulterà **VERA**; 4) Viene anche chiamato **costrutto di iterazione o ciclo "con controllo in testa"**	1) La condizione viene controllata **dopo** avere seguito il blocco di istruzioni <B1>; 2) Esegue il blocco di istruzioni <B1> **almeno una volta**; 3) Esegue il blocco di istruzioni <B1> fino a che la condizione in coda risulterà **FALSA**; 4) Viene anche chiamato **costrutto di iterazione o ciclo "con controllo in coda"**

OSS: In alcuni casi potremmo conoscere FIN DALL'INIZIO quante volte deve essere ripetuto un ciclo. Quando si conosce FIN DALL'INIZIO quante volte deve essere ripetuta un'azione è possibile utilizzare PER MAGGIORE SEMPLICITA' DI SCRITTURA il costrutto iterativo ENUMERATIVO o determinato o indicizzato.

Pseudocodifica *istruzione di "Iterazione ENUMERATIVA o DETERMINATA o INDICIZZATA":*

```
.....
PER <indice> ← <inizio> [INDIETRO] A <fine> ESEGUI
      <B1>
      INCREMENTA <indice>
      [DECREMENTA <indice>]
FINE PER
.....
```

oppure in modo equivalente:

```
.....
PER <indice> ← <inizio> [INDIETRO] A <fine> ESEGUI
      <B1>
      <indice> ← <indice> + 1
      [<indice> ← <indice> - 1]
FINE PER
.....
```

N.B. Tale tipo di costrutto iterativo viene utilizzato ESCLUSIVAMENTE quando si conoscono a priori il numero di iterazioni richieste.

La <condizione> è SOTTOINTESA ed è esprimibile con il seguente enunciato composto dell'Algebra di Boole:

- **<indice> ≤ <fine>** se il **PER** è **incrementale** (o **CRESCENTE**)
- **<indice> ≥ <inizio>** se il **PER** è **decrementale** (o **DECRESCENTE**)

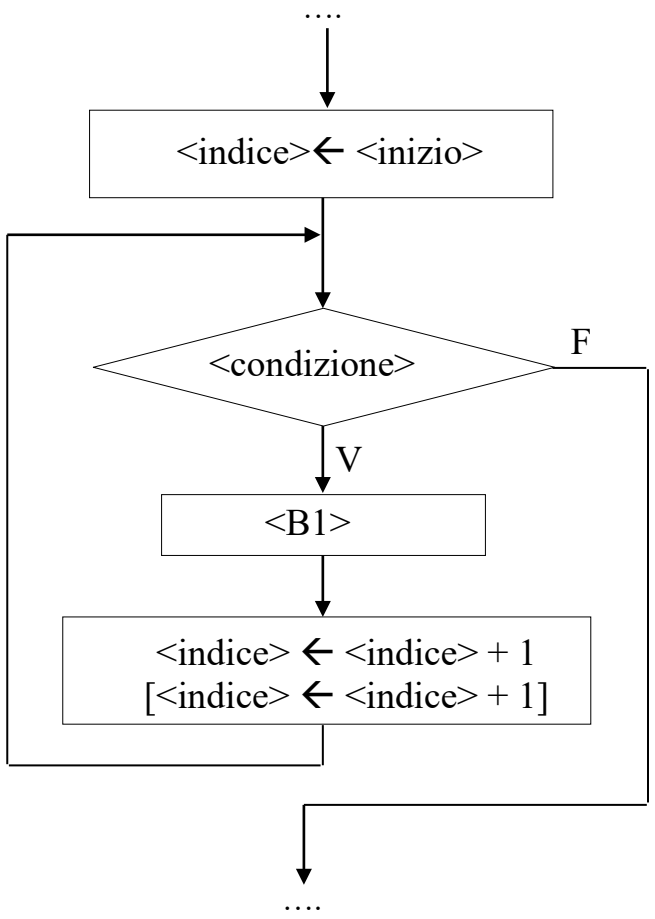
essendo sempre <B1> un qualsiasi **blocco semplice** o un **blocco composto** di istruzioni**.**

Se la condizione SOTTOINTESA è VERA si esegue il blocco di istruzioni <B1>

Se la condizione SOTTINTESA è FALSA si arresta il processo iterativo.

Se la condizione SOTTINTESA è inizialmente FALSA il ciclo PER non viene mai eseguito

***Flow-chart** istruzione di "Iterazione ENUMERATIVA o DETERMINATA o INDICIZZATA":*



E' possibile provare che tale tipo di costrutto equivale ad uno iterativo con controllo in testa e che quest'ultimo equivale ad un **costrutto di selezione + costrutto iterativo** con controllo in coda

Esempio esplicativo

PER i ← 1 **A** 5 **ESEGUI**
    <B1>
    i ← i + 1
FINE PER

Costrutto iterazione determinato o indicizzato

equivale ⟺

i ← 1

MENTRE (i ≤ 5) **ESEGUI**
    <B1>
    i ← i + 1
FINE MENTRE

Costrutto iterazione con controllo in testa o per vero

equivale ⟺

i ← 1

SE (i ≤ 5)
    ALLORA
        RIPETI
            <B1>
            i ← i + 1
        FINCHE' NOT (i ≤ 5)
FINE SE

Costrutto iterazione con controllo in coda o per falso

N.B. Ovviamente non tutte le iterative di tipo MENTRE o RIPETI possono essere rappresentate in modo equivalente utilizzando una istruzione di tipo PER.

Esempio
RIPETI
    Leggi(a)
FINCHE' (a > 0)

In questo esempio il numero di iterazioni (cicli) dipenderà dall'immissione o meno da parte dell'utente di un valore maggiore di 0.
N.B. Non è possibile scrivere un'istruzione PER equivalente a questa istruzione iterativa di tipo RIPETI, in quanto non è noto a priori il numero di cicli

PROGRAMMAZIONE STRUTTURATA

Def: Con il termine **programmazione strutturata** si intende la programmazione che considera l'algoritmo come un insieme di blocchi di azioni o istruzioni ognuno fornito di un solo ingresso e di una sola uscita (blocchi di istruzioni ONE IN - ONE OUT).

Ciascun blocco è isolato dagli altri nel senso che **non è possibile saltare dall'interno di un blocco all'interno di un altro** (programmazione a salti).

Due matematici italiani Corrado **BOHM** e Giuseppe **IACOPINI** nel 1966 formularono un importantissimo teorema che non dimostreremo che affermava:

Teorema di BOHM-JACOPINI**: Un algoritmo scritto secondo le regole della programmazione a salti (ossia non strutturato) per quanto complesso può SEMPRE essere trasformato in un algoritmo ad esso equivalente che utilizzi esclusivamente le tre strutture di controllo fondamentali: sequenza, selezione ed iterazione (ossia strutturato).**

N.b. In realtà si può fare vedere che bastano la struttura di sequenza, la struttura di selezione binaria e la struttura di iterazione con controllo in coda (istruzione ripetitiva RIPETI)

Il fondamentale teorema di **BOHM-JACOPINI** ci assicura che le tre strutture di controllo fondamentali formano un insieme di strutture completo, cioè tramite le quali si possono descrivere tutti gli algoritmi.

Conseguenza del Teorema di BOHM-JACOPINI: **Ogni algoritmo può essere espresso con le sole tre strutture di controllo fondamentali sequenza, selezione e iterazione**

Il **primo requisito** impone che per la realizzazione dell'algoritmo e del programma vengano impiegate solo tre strutture di controllo fondamentali.

Il **secondo requisito** della programmazione strutturata impone che ogni struttura di controllo dell'algoritmo, immaginata mediante il formalismo degli schemi a blocchi, debba essere un blocco con una sola freccia in entrata ed una sola in uscita.

Ogni blocco interno ad una struttura di controllo non è detto che sia semplice (singola istruzione) ma può essere a sua volta una struttura.

Imporre che una soluzione algoritmica di un problema sia scritta seguendo questi dettami non è un puro esercizio di programmazione ma porta dei vantaggi diretti fra cui:

Vantaggi derivanti dal Teorema di BOHM-JACOPINI:

- maggiore facilità di uso di tecniche di progettazione quali la metodologia top-down;
- algoritmi più leggibili;
- maggiore facilità di individuazione degli errori.

N.B. E' da notare che la programmazione strutturata non è la più moderna metodologia per scrivere programmi, anzi, forse, è da considerarsi come una delle prime: ciononostante è riconosciuta ancora oggi come una delle fondamentali.

I concetti della programmazione strutturata sono stati subito impiegati nell'industria e oggigiorno sono stati recepiti da tutti i linguaggi di programmazione moderni.