# I.F.T.S. 2020 - 2021

# DISPENSE DIDATTICHE

# Modulo n°: 7

# Titolo modulo: FONDAMENTI DI PROGRAMMAZIONE E INGEGNERIA DEL SOFTWARE

# Docente: Claudio Rossi

# Coordinatrice del Corso: Sara Forlivesi

Fondazione En.A.I.P. S. Zavatta Rimini
Viale Valturio, 4 47923 Rimini
Tel. 0541.367100 – fax. 0541.784001
www.enaiprimini.org; e-mail: info@enaiprimini.org

# Cos’è un problema

Esistono molti tipi di problemi, non tutti si prestano ad essere risolti mediante l’uso di computer.
Vediamo alcuni tipi di problema:

**Problema aritmetico**

3521+
4768=
?

Qual è il risultato dell’operazione?

**Problema criptoaritmetico**

MARE +
SOLE =
FREDDO

Quali cifre aritmetiche devono essere sostituite ai caratteri per far sì che la somma abbia senso?

**Problema non matematico**

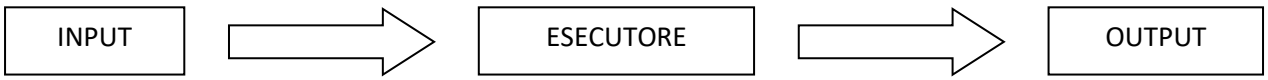
Che cravatta abbinare ad una giacca blu e ad un pantalone grigio?

**Problema di programmazione lineare**

Un commesso viaggiatore deve visitare un certo numero di città almeno una volta e conosce la distanza tra le varie città.
Com'è possibile determinare il percorso più breve?
Il numero dei percorsi possibili dipende da n! per cui nel caso di 20 città i percorsi sono già più svariati miliardi di miliardi

**Risolvere un problema significa:**

Individuare e rappresentare un elenco di istruzioni che, interpretate da un esecutore, conducono da un insieme di informazioni iniziali ad un insieme di informazioni finali soddisfacenti un criterio di verifica.
Il risultato è costituito dall'insieme dei dati finali.
La soluzione di un problema è la sequenza di istruzioni che, a partire dai dati di ingresso, permettono di ottenere i risultati.




**Come si risolve un problema**

Per risolvere un problema è necessario seguire un procedimento del seguente tipo:

- Definire il problema
- Individuare un criterio di verifica
- Descrivere il problema mediante un modello matematico
- Approssimare il modello mediante un metodo numerico
- Sviluppare l'algoritmo risolutivo ed applicarlo ad uno specifico ambiente

**Un problema è ben formulato se:**

- Il criterio di verifica è univoco ed applicabile
- l'insieme dei dati iniziali è completo.

Inoltre non deve essere evidente in partenza la mancanza di soluzioni.
La risoluzione di un problema è fortemente condizionata dall'esecutore disponibile, cioè dall'insieme delle istruzioni che esso è in grado di interpretare
Il **processo risolutivo** è un'azione composta da una sequenza di azioni elementari (**passi**)

**La soluzione di un problema è effettiva per un esecutore se:**

- l’esecutore è in grado di interpretare la soluzione
- l’esecutore è in grado di compiere le azioni richieste in un tempo finito

**Un esecutore è caratterizzato da:**

- il linguaggio che è in grado di interpretare
- l’insieme di azioni che è in grado di compiere
- l’insieme delle regole che associano ad ogni costrutto linguistico le relative azioni

**concetto di algoritmo (definizione intuitiva)**

**L'algoritmo è una sequenza finita di istruzioni** univocamente interpretabili da un esecutore che individua una sequenza finita di operazioni elementari, atte a fornire i risultati di una classe di problemi, per qualsiasi valore dei dati in ingresso.

- **sequenza**: insieme in cui è stabilita una relazione d'ordine
- **istruzioni**: comandi interpretabili da un esecutore che li trasforma in azioni

- **non ambigue**: ciascuna istruzione è trasformata in un insieme di operazioni certamente definibile, noto il contesto in cui si opera
- **operazioni**: azioni elaborative ottenute dall'interpretazione delle istruzioni
- **problema**: quesito che attende una risposta
- **classe di problemi**: insieme di problemi della medesima tipologia

I concetti di **algoritmo** ed **esecutore** sono strettamente connessi.
Una sequenza di istruzioni costituisce o meno un algoritmo a seconda dell'esecutore su cui lavora.
La **progettazione** e la **descrizione** di un algoritmo dipendono dalle capacità dell'esecutore.
**L'esecutore** interpreta ed esegue nel medesimo modo una stessa istruzione a parità di condizioni al contorno.
Se l'algoritmo può anche **non terminare,** si parla di **metodi computazionali**: sono ammesse esecuzioni con un numero infinito di passi.

# La macchina di Turing

Il matematico inglese **Alan Turing** ha formalizzato la nozione di algoritmo descrivendo in maniera formale un esecutore, detto **macchina di Turing** (MdT)**.**
La **MdT** è una Terna

$$Z=(A,Q,P)$$

dove:
**A** è l'alfabeto finito dei simboli leggibili/scrivibili
**Q** è l'insieme finito degli stati dell'organo di controllo
**P** è l'insieme finito delle quintuple che ne descrivono il funzionamento.
Cioè, dato

- uno stato iniziale
- un carattere letto

siamo in grado di stabilire

- lo stato successivo
- il carattere scritto
- la direzione della testina di lettura/scrittura

Avendo la nozione di Macchina di Turing possiamo rispondere alle seguenti domande:

- Quando un problema è risolvibile?
- Quando una funzione è effettivamente calcolabile?

**Tesi di CHURCH**
Ogni funzione calcolabile è Turing-calcolabile cioè può essere eseguita su una macchina di Turing.
In definitiva i problemi possono essere

- **solvibili**: risolvibili da MdT o da procedure effettive
- **insolvibili**

## Caratteristiche che deve possedere un esecutore

- capacità di comunicare con l'esterno (operazioni di I/O)
- capacità di conservare informazioni (memorizzazione di dati ed istruzioni)
- capacità di effettuare operazioni logico-aritmetiche
- capacità di interpretare ed eseguire un algoritmo, decodificando ciascuna istruzione ed individuando quale sarà la prossima istruzione da eseguire.

Gli algoritmi applicati ad un calcolatore vengono detti **programmi**
**Esempio l'algoritmo per la determinazione del massimo comune divisore di due numeri**

- scomporre i due numeri in fattori primi
- moltiplicare tra loro i fattori comuni presi una sola volta con il minimo esponente.

Ciò comporta che **l'esecutore sappia scomporre** i numeri in fattori primi, sappia individuare i fattori comuni con il minimo esponente, sappia effettuare la moltiplicazione.

## Algoritmo di Euclide

**Problema**
Determinare il massimo comune divisore di due numeri naturali diversi da 0.
**Ipotesi**
L'esecutore è in grado di:

- effettuare la divisione,
- individuare il resto della divisione,
- effettuare operazioni logiche,

- stabilire quale è la prossima istruzione da eseguire.

Chiamiamo A Il primo numero, B il secondo ed r il resto.
L'algoritmo di Euclide è il seguente:

1) dividi il primo numero per il secondo;
2) chiama r il resto della divisione.
3) se r = 0 il secondo numero è il MCD cercato
4) altrimenti
5) primo numero ← secondo numero
6) secondo numero ← r
7) ritorna alla prima istruzione

Scriviamo l'algoritmo di Euclide con un linguaggio detto di pseudoprogrammazione, ossia un linguaggio a metà strada fra quello umano e quelli di programmazione:

```
ripeti
        r = A mod B
        se r = 0 allora
                scrivi (" il MCD è", B)
        altrimenti
                A = B
                B = r
        fine se
finché r = 0 ;
```

# Elementi di programmazione imperativa

## Caratteristiche generali di un programma

- Un programma permette di acquisire dati (Input), elaborarli, produrre risultati (Output).
- Un programma in esecuzione necessita di un processore, in grado di interpretare le istruzioni e di eseguirle, e di una memoria in grado di conservare dati ed istruzioni per il tempo necessario all'esecuzione.
- I programmi sono costituiti da istruzioni, ciascuna con una propria sintassi ed una propria semantica, cui corrisponde l'azione da eseguire.
- I dati possono essere di vari tipi.
- Ciascun tipo di dato è individuato da un nome, dall'insieme dei valori che può assumere e dalle operazioni possibili.
- Ad esempio *giallo*, *rosso*, *blu* sono dei valori del tipo *colore* e *Mescola* (a,b) è una delle operazioni possibili.

## Costanti e variabili

- I dati utilizzati in un programma possono essere costanti o variabili.
- Una costante è un numero, una scritta, oppure un nome simbolico il cui valore non varia nel corso dell'esecuzione del programma.
- Una variabile è un nome simbolico cui è associato un tipo ed un valore variabile nel corso del programma.
- Variabili e costanti sono memorizzate in memoria centrale durante l'esecuzione del programma e possono occupare una o più locazioni di memoria a seconda della loro tipologia.

**Esempio di dichiarazione di variabili**
Var i:integer; La variabile i è intera e può assumere i valori interi compresi tra -32768 e 32767. Occupa due locazioni di memoria da 8 bit.
Var Cognome:string[20]; La variabile Cognome è una stringa di 20 caratteri che possono assumere qualsiasi valore tra le 256 configurazioni del codice ASCII. Occupa 20 locazioni di memoria.
Var x,y:real; Le due variabili x e y sono di tipo reale, cioè possono assumere valori interi o decimali. Occupano una quantità di byte variabile a seconda del linguaggio di programmazione utilizzato, memorizzano mantissa ed esponente con una notazione detta floating point.

## Assegnazione

L'operazione di assegnazione permette di memorizzare nella variabile presente nel primo membro il valore dell'espressione presente nel secondo membro, ottenuto sostituendo a ciascuna variabile eventualmente presente il valore corrispondente ed effettuando le operazioni necessarie.
**Esempio**
I =1; I vale 1, cioè in memoria centrale le locazioni corrispondenti alla variabile I contengono il valore 1.
I =I+1; I valeva 1; dopo questa assegnazione I vale 2;
J =5; J vale 5;
K =(I+J)*(I+1); K vale (2+5)*(2+1), cioè 21.

## Scomposizione di un problema

Per risolvere un problema è opportuno scomporlo in sottoproblemi secondo la metodologia top-down, le procedure e le funzioni sono utili a questo scopo.
Procedure e funzioni sono sottoprogrammi che possono essere richiamati ricevendo parametri di input, elaborando i dati e restituendo, al termine dell'esecuzione, parametri di output al programma chiamante.

La scomposizione può avvenire secondo tre modalità di base:

1) sequenziale
2) condizionale
3) iterativa.

## Scomposizione sequenziale

Il problema viene scomposto in una sequenza di sottoproblemi, le istruzioni corrispondenti vengono eseguite in ordine sequenziale, una dopo l'altra.

**Esempio**

...
Disegna_testa
Disegna_corpo
Disegna_arti
....

La realtà esterna, l'evolversi dei dati non modifica l'ordine di esecuzione delle istruzioni.

**Esempio**

Calcolare la somma di due numeri interi.

1) Leggi a
2) Leggi b
3) Somma a + b
4) Scrivi Somma

## Scomposizione condizionale

Il problema viene scomposto in due sottoproblemi alternativi tra loro.
Per stabilire quale dei due percorsi risolutivi bisogna seguire ci si avvale di proposizioni, cioè di espressioni linguistiche delle quali si può valutare la verità, e più specificamente di proposizioni variabili (o predicati) cioè di affermazioni la cui verità dipende dai valori delle variabili contenute. Sinteticamente si parla di condizioni.

**Esempio**

leggi(a);
if a >= 0 then scrivi("a è positivo")
else scrivi("a è negativo");

## Scomposizione iterativa (ripetitiva)

Il problema è scomposto in un insieme di operazioni che possono essere ripetute più volte (ciclo), sino al verificarsi di una certa condizione (predicato di controllo).
In quasi tutti i linguaggi di programmazione sono disponibili tre costrutti sintattici per eseguire ciclicamente un blocco strutturato di istruzioni:
While...,
For..,
Repeat.. Until.

# Il costrutto while

Prevede l'esecuzione ciclica di un insieme (blocco strutturato) di istruzioni se la condizione (guardia) è vera.
La condizione può anche essere composta.
Se la condizione è falsa il programma salta il blocco di istruzioni contenuto nel ciclo while e continua con la prima istruzione successiva.
Il while necessita di alcuni elementi minimi per poter funzionare correttamente:

1) la presenza di una condizione che contenga almeno una variabile di controllo (senza variabili il ciclo continua all'infinito o non inizia mai).
2) la memorizzazione di un valore nella variabile di controllo prima di testare la condizione
3) la modifica del valore della variabile di controllo all'interno del ciclo, per evitare che il ciclo continui all'infinito.

Il costrutto while è molto flessibile in quanto non pone limiti al numero di cicli eseguibili, al tipo di condizione, al numero

delle variabili di controllo ed alle relative modalità di modifica.
È particolarmente adatto a quelle tipologie di problemi in cui non si conosce a priori il numero di cicli che bisogna eseguire.
**Esempio**
Somma di N numeri positivi immessi da tastiera.
(Zero per terminare)

```
Somma = 0;
leggi(Numero);
while Numero <> 0 do
        Somma = Somma + Numero;
        leggi(Numero);
end while;
scrivi(Somma);
```

# Il costrutto repeat-until

È molto simile al costrutto sintattico while ma la condizione viene valutata alla fine del ciclo, per cui le istruzioni cicliche vanno eseguite almeno una volta.
Al contrario del costrutto while, se il predicato di controllo è vero si esce dal ciclo, altrimenti si continua l'iterazione.
Esempio

```
Somma = 0;
repeat
        leggi(Numero);
        Somma = Somma + Numero;
until Numero <> 0;
scrivi(Somma);
```

# Il costrutto for

Questo costrutto consente di effettuare un determinato numero di volte un blocco di istruzioni (ciclo).
È adatto a situazioni in cui è noto a priori il numero di cicli da eseguire.
**Esempio**

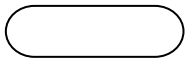
```
for i = 1 to n do
        leggi(Num);
        Somma = Somma + Num;
end;
```

# Istruzioni di input e di output

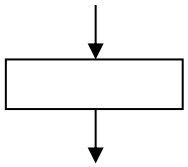
Permettono di acquisire dati o di trasferirli all'esterno.
La sintassi dipende dalle periferiche che intendiamo utilizzare (tastiera, monitor, stampante).

# Diagrammi a blocchi

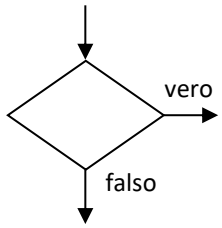
Per rappresentare graficamente un algoritmo è stato standardizzato un metodo chiamato **diagramma a blocchi**.
In un diagramma a blocchi si rappresenta ogni istruzione con un elemento geometrico e si rappresenta il flusso delle operazioni (ossia l'ordine in queste vengono eseguite) collegando gli elementi con degli archi orientati (linee che terminano con una freccia).



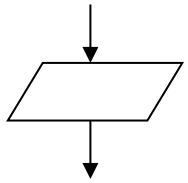
inizio o fine dell'esecuzione



calcolo o assegnamento

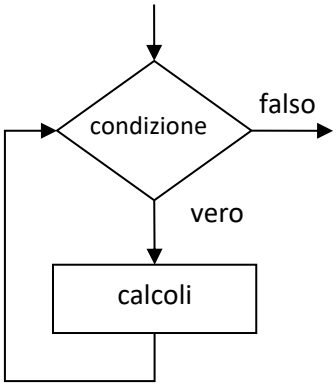



istruzione di controllo ed esecuzione alternativa



istruzione di lettura o scrittura dei dati

Nei diagrammi a blocchi non si ha la fase di dichiarazione delle variabili, infatti manca l'equivalente della istruzione di dichiarazione, questo non è importante dato che i diagrammi servono a visualizzare il **flusso** dell'esecuzione (infatti si chiamano anche diagrammi di flusso) e non a rappresentare esattamente un programma.
Non esiste nessun blocco particolare per indicare le strutture di iterazione; si usano semplicemente dei blocchi di selezione in testa o in coda al gruppo di istruzioni facenti parte del ciclo.
Esempio: ciclo while



# Metodologia di scomposizione top-down

Per risolvere un problema complesso è utile individuarne i diversi aspetti in maniera generale prima di passare allo sviluppo di dettaglio.
È particolarmente vantaggioso riuscire a scomporre un problema in sottoproblemi, eventualmente a loro volta scomponibili, in modo da poter affidare le varie fasi della soluzione anche a gruppi di lavoro diversi, questo tipo di scomposizione prende il nome di top-down.
La metodologia top-down richiede particolare attenzione nell'individuazione dei sottoproblemi, nella progettazione dei sottoprogrammi che li risolvono e nei rapporti tra sottoprogrammi.
Le procedure e le funzioni della maggior parte dei linguaggi di programmazione permettono di realizzare tali sottoprogrammi.
Ogni sottoprogramma è caratterizzato da una fase elaborativa che permette di ottenere gli output desiderati a partire dagli input inseriti.
**Esempio**
Vogliamo calcolare l'area di figure geometriche, costituite da un tipo (cerchio, quadrato,...) e dalle misure dei lati.
La figura geometrica va vista come un insieme di parti, un rettangolo può essere visto, ad esempio, come un insieme di elementi costituito da "tipo=rettangolo; base=20; altezza=12".

```
Program Calcola_Area
...
for i =1 to n do
Leggi_Dati(tipo, lati);
Calcola_Area(tipo, lati, area);
Stampa_Area(area)
...
end.
```

**Osservazioni**
Alcune parole riservate (come while, end, for) assumono un significato fisso definito nel linguaggio.
Altre parole (Leggi_Dati, Calcola_Area, tipo, lati, area) assumono il significato che l'utente definisce nel corso del programma specifico.
tipo, lati, area sono variabili, contengono un valore che può variare nel corso del programma.
Alcune istruzioni vanno eseguite in sequenza, una dopo l'altra, altre istruzioni vanno eseguite ripetutamente fino a quando non si verifica una determinata condizione

# Programmi e sottoprogrammi

I programmi possono essere suddivisi in sottoprogrammi.
Ciascun sottoprogramma può essere richiamato dal programma principale con il proprio nome.
Ogni programma può chiamare al suo interno uno o più sottoprogrammi cui sono delegate particolari funzioni.

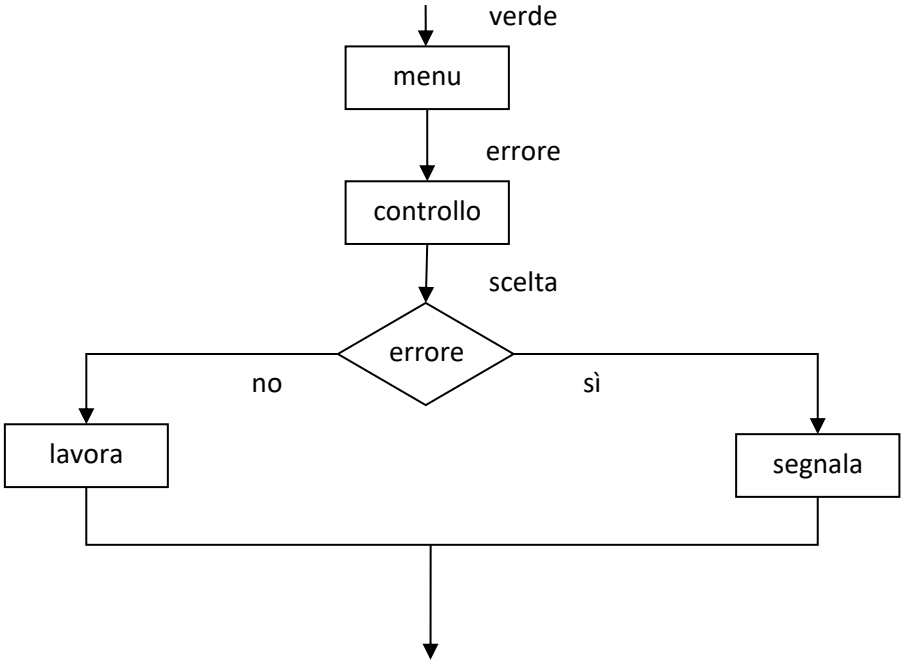
```
...
acquisisci(a,b);
elabora(a,b,c);
visualizza(c);
....
```

Ciò aumenta notevolmente la flessibilità e la leggibilità del programma.
L'utilizzo ripetuto di un sottoprogramma evita la ripetizione delle singole istruzioni che lo compongono semplificando la struttura del programma chiamante e la manutenzione del codice (eventuali errori nella sequenza di istruzioni si devono correggere una sola volta).
Il sottoprogramma si comporta come una scatola nera che si relaziona con il programma chiamante tramite i parametri, riceve in input dal programma chiamante un insieme di valori costanti o variabili, i parametri di input.
Il sottoprogramma esegue le sue fasi elaborative e può assegnare valori opportuni ad un insieme di variabili, i parametri di output, che vengono rese disponibili al programma chiamante.
La scomposizione di un programma in sottoprogrammi è molto utile nella progettazione perché permette di razionalizzare il lavoro sviluppandolo in tempi diversi ed assegnandolo anche a team distinti, a condizione che siano ben definiti i significati dei parametri di input e di output.

## Parametri

I parametri sono la maniera di comunicare del sottoprogramma con il mondo esterno.
Alla fine della sua esecuzione il sottoprogramma trasferisce il controllo al programma chiamante.
**Esempio**

Programma principale
...
Menu (Verde,Scelta);
Controllo(Errore,Scelta);
If Errore=true then Segnala(Errore)
else Lavora (Scelta);
...




## Procedure e funzioni

In quasi tutti i linguaggi di programmazione sono presenti due tipi di sottoprogrammi: procedure e funzioni.
Le procedure comunicano con il programma chiamante esclusivamente tramite i parametri.
**Esempio di utilizzo di procedure**
Visualizza(a,b);
Anche le funzioni comunicano tramite parametri ma possono anche associare al proprio nome un valore che può essere direttamente utilizzato nel programma chiamante.
**Esempio di utilizzo di funzione**
a =random(n);
scrivi(random(n));

# Ricorsione

Si chiamano *ricorsive* le procedure o funzioni che, direttamente o indirettamente, richiamano se stesse; si dice anche che si sta usando la tecnica della ricorsione.
Alcuni problemi di natura matematica si prestano all'implementazione ricorsiva perché sono per loro natura ricorsivi, un ottimo esempio è la funzione fattoriale.
Ricordiamo che il fattoriale di un numero n si indica con n! e significa n*(n-1)*(n-2)*....*1; per convenzione si pone 0! = 1
**Esempio: algoritmo per il calcolo del fattoriale**
È facile scrivere l'algoritmo per calcolare fatt(n), cioè n!, usando un "for":

```
Leggi(n);
fattoriale = 1;
for i = 1 to n
        fattoriale = fattoriale * i
end
```

Vediamo ora come calcolare n! usando la ricorsione
Per risolvere il problema consideriamo la seguente eguaglianza: (n+1)! = (n+1) * n!
Questa uguaglianza può essere utilizzata per dare una definizione ricorsiva della funzione fattoriale:

```
        [base]      fatt(0) = 1
        [passo]     fatt(n+1) = (n+1) * fatt(n)
```

Che possiamo tradurre nella seguente funzione scritta in pseudocodice

```
Funzione fatt (n : integer) : integer
        if n=0 then
                return 1
        else
                return (n * fatt (n-1))
        end
end
```

Per capire come procede l'esecuzione di una chiamata, supponiamo che in un programma ci sia l'istruzione scrivi( fatt (3)).
L'esecuzione procede così:
fatt(3) → 3 * fatt(2) → 3 * 2 * fatt(1) → 3*2*1* fatt(0) → 3*2*1*1 → scrivi 6
Il fattoriale è un esempio di ricorsione diretta: nel corpo della funzione fatt compare una chiamata a fatt.
Il punto chiave è che il nome "fatt" è noto all'interno del corpo della funzione fatt e si riferisce alla funzione stessa.

## Ricorsione indiretta e incrociata

Finora abbiamo parlato di ricorsione diretta: una funzione o procedura contiene, nel corpo, una o più chiamate a se stessa.
Si può anche avere una ricorsione indiretta:
la procedura P chiama direttamente la procedura Q, che chiama direttamente la procedura R,....., che chiama la procedura P.
È anche possibile la ricorsione incrociata, nel caso più semplice: P chiama Q e Q chiama P.

# Dati e codifica

I dati vengono rappresentati da sequenze di simboli, scelti da un insieme finito, detto alfabeto.
Ad ogni alfabeto va associato un insieme di regole di composizione
Esempio: rappresentazione di un numero nel sistema di numerazione decimale

153 significa

1x100 + 5x10 + 3x1

in quanto il sistema di numerazione decimale è posizionale.

## Alfabeto binario

I computer utilizzano segnali elettrici per rappresentare le informazioni, questo porta come logica conseguenza l'utilizzo della base 2 come base di numerazione.
L'elemento base è detto **bit** e può assumere due soli valori: 0 oppure 1
0 è un messaggio di un bit
1 è un messaggio di un bit
0100 è un messaggio di quattro bit
01001101 è un messaggio di 8 bit (byte)
Con k bit si possono ottenere $2^k$ configurazioni diverse, ad esempio, con 4 bit si possono ottenere 16 configurazioni diverse.
Il numero di simboli (bit) necessario per codificare un messaggio in alfabeto binario è maggiore rispetto ad un alfabeto con più simboli.
Ad esempio
$14_{10}$ richiede due cifre decimali
il corrispondente binario 1110 richiede quattro cifre binarie (bit).
In generale il numero di bit necessario a codificare un insieme costituito da n diversi messaggi è dato da
$b = \log_2 n$
Se il valore ottenuto non è un numero intero si arrotonda per eccesso.

## Elaboratori ed alfabeto binario

I computer trattano l'informazione in maniera discreta, cioè considerano significativi solo alcuni valori delle grandezze fisiche trattate, quindi quando si digitalizza un'informazione analogica (suono, immagine,...) si ottiene sempre una approssimazione.
Il byte è significativo perché ad esso è comunemente associato un carattere nei codici di uso più comune, come il codice ASCII ed il codice EBCDIC. Ad esempio, nel codice ASCII al carattere *a* corrisponde 0110 0001

### Codifica dei dati non numerici

Un singolo bit consente di distinguere i dati in due sottoinsiemi.
Più bit consentono di distinguere i dati in più sottoinsiemi.
Esempio
S = cuori, quadri, fiori, picche
La cardinalità (numero di elementi) di S è 4
Per rappresentare S sono necessari almeno 2 bit ($2^2=4$)
Ad esempio possiamo codificare le informazioni nella seguente maniera:

| | primo bit | secondo bit |
|---|---|---|
| cuori | 0 | 0 |
| quadri | 0 | 1 |
| fiori | 1 | 0 |
| picche | 1 | 1 |

Codice ASCII (American Standard Code For Information Interchange)
Utilizza 8 bit, quindi consente di rappresentare $2^8=256$ caratteri
In particolare rappresenta:

- 26 lettere dell'alfabeto maiuscole
- 26 lettere dell'alfabeto minuscole
- 10 cifre decimali

- 30 simboli di interpunzione e di uso corrente
- caratteri speciali.

Codice EBCDIC
Anch'esso utilizza 8 bit
È stato sviluppato da IBM
Codice UNICODE
Utilizza 16 bit, quindi è in grado di rappresentare $2^{16}$=65536 diversi caratteri.
È utile per rappresentare diversi alfabeti (arabo, greco, matematico, grafico).
I primi 128 caratteri coincidono con quelli del codice ASCII ( i 7 bit meno significativi).

## La codifica delle istruzioni

Le istruzioni sono interpretate ed eseguite da un elaboratore.
Il linguaggio macchina è direttamente interpretabile dall'elaboratore ed è costituito da istruzioni basate su un alfabeto binario.
**Esempio**
01100000 significa ADD (somma)
Ad ogni istruzione è associato un codice univoco, detto codice operativo.
Spesso le istruzioni sono costituite da un codice operativo e da operandi.
**Esempio**
Se voglio sommare il contenuto due locazioni di memoria l'istruzione sarà costituita da :

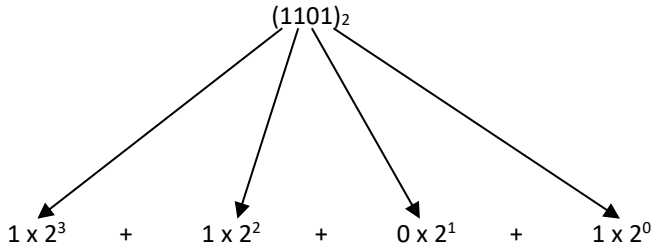
- codice operativo;
- indirizzo del primo operando;
- indirizzo del secondo operando.

Ogni microprocessore ha il proprio set di istruzioni e proprie regole di codifica, questo significa che un programma scritto per un particolare tipo di processore non può funzionare su processori che hanno set di istruzioni diversi.

## La codifica di dati numerici

Per codificare i dati numerici servono tre cose:

- una base di riferimento (decimale, binario, esadecimale, ottale);
- un insieme di regole di codifica;
- uno o più sistemi di numerazione posizionali.




Per gli esseri umani la base di riferimento è decimale, per gli elaboratori è quella binaria.
I codici esadecimale ed ottale si usano perché rendono sinteticamente la notazione utilizzata dagli elaboratori (ogni cifra rappresenta rispettivamente 2B e 1B).

### Conversione da binario a decimale

Per convertire un numero binario in uno decimale, è sufficiente moltiplicare ogni cifra per la base elevata alla posizione occupata dalla cifra e sommare i valori così ottenuti.
**Esempio**
Cerchiamo il valore decimale del numero binario 1101

| 1 | 1 | 0 | 1 |
|---|---|---|---|
| $2^3$ | $2^2$ | $2^1$ | $2^0$ |

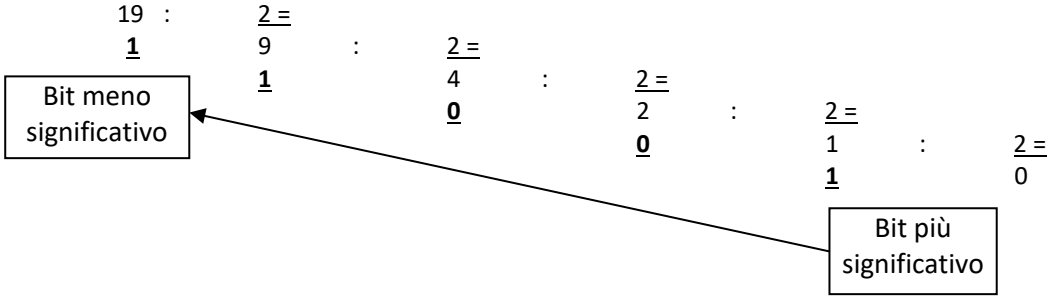
$$1*2^3+1*2^2+1*2^1+1*2^0 = 13$$
$$1101_2=13_{10}$$

## Conversione da decimale a binario

Per convertire un numero decimale in uno binario, bisogna dividere il numero per 2, prendere il risultato ottenuto e dividerlo per 2, continuare con il procedimento fino a quando il risultato della divisione è zero.
Il numero binario cercato si ottiene leggendo i resti delle divisioni.

**Esempio**

Cerchiamo la rappresentazione binaria del numero decimale 19




$19_{10} = 10011_2$
Infatti
$1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 19$

## Codifica esadecimale

È una notazione sintetica comoda per rappresentare i byte, ciascuna cifra esadecimale corrisponde ad un gruppo di quattro bit, quindi due cifre rappresentano esattamente un byte.
L'alfabeto esadecimale è rappresentato nella seguente tabella

| cifra esadecimale | valore decimale |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |
| A | 10 |
| B | 11 |
| C | 12 |
| D | 13 |
| E | 14 |
| F | 15 |

**Esempio**
$237_{10} = 11111100_2 = FC_{16}$
$125_{10} = 01111101_2 = 7D_{16}$

## Codifica ottale

Anche questa è una notazione che viene utilizzata perché è sintetica, ciascuna cifra ottale corrisponde ad un gruppo di 3 cifre binarie

# Codifica binaria di numeri interi con segno

Finora abbiamo visto solamente conversione di numeri postivi, ovviamente nell'utilizzo pratico è necessario codificare anche numeri negativi.
In binario è possibile codificare segno e modulo, il bit più significativo è riservato al segno

- 0 segno positivo
- 1 segno negativo

gli altri bit sono riservati al modulo (valore del numero senza segno).

**Esempio**

0100 significa +4

1100 significa –4

Utilizzando questa rappresentazione, con n bit si possono rappresentare i numeri da

$$-(2^{n-1} - 1) \text{ a } (2^{n-1} - 1)$$

NB: zero è rappresentato due volte (+0 e -0).

## Codifica binaria di numeri razionali

In questo caso siamo di fronte ad un problema molto grande: il numero di cifre decimali può essere superiore al numero di cifre rappresentabili o addirittura infinito!

Per cercare di ovviare al problema vengono memorizzate solo le cifre più significative.

Un altro problema è la necessità di rappresentare numeri con valore assoluto molto grande o molto piccolo, per risolvere questo problema si usa la rappresentazione a virgola mobile (notazione scientifica).

$$\pm \text{ mantissa} * \text{base}^{\text{esponente}}$$

**Esempio**

750 si scrive come $0{,}75 * 10^3$

Esiste uno standard di codifica (altrimenti sarebbe delirante cercare di far comunicare fra loro sistemi diversi)

Esempio: rappresentazione con 64 bit

| 1 bit<br>segno | 11 bit<br>esponente | 52 bit<br>mantissa |
|---|---|---|

Problemi di questa soluzione

Se si vuole rappresentare un numero maggiore del massimo rappresentabile si verifica un errore detto di overflow.

Se si vuole rappresentare un numero minore del minimo rappresentabile si verifica una situazione detta di underflow.

In tutti gli altri casi la rappresentazione è possibile ma può verificarsi la necessità di arrotondamenti, ogni arrotondamento introduce un errore di cui va valutata la tollerabilità.

# I dati

Quando si lavora con i linguaggi di programmazione, le variabili sono caratterizzate da nome, valore e tipo.
**Esempio**
La dichiarazione var Contatore: integer;
e l'istruzione Contatore = 1;
indicano una variabile di nome Contatore, di tipo intero, contenente il valore 1.
Alcuni tipi di dato sono predefiniti, altri possono essere definiti dall'utente.
Un tipo di dato è caratterizzato da:

- un nome (ad esempio integer)
- un insieme dei valori ammissibili ( -32768 < x < 32767)
- un insieme delle operazioni possibili ( -, +, /, div, * ).

Vediamo nel seguito i tipi di dato più utilizzati.

# Il tipo boolean

Può assumere due valori: true oppure false
I principali operatori booleani sono:

| **AND** | **true** | **false** |
|---|---|---|
| **true** | true | false |
| **false** | false | false |

| **OR** | **true** | **false** |
|---|---|---|
| **true** | true | true |
| **false** | true | false |

| **NOT** | |
|---|---|
| **true** | false |
| **false** | true |

**Esempio**
if (x > 0) and (x < 20) then....
è vera per i valori di x interni all'intervallo di estremi 0 e 20 (estremi esclusi).
if ( x < 10 ) or (x > 50 ) then..
è vera per i valori di x esterni all'intervallo di estremi 10 e 50 (estremi esclusi)..
if (i <= n) and (ok = false) then...
è vera se si verificano entrambe le condizioni, cioè se i <= n e ok = false.

# Tipi strutturati

Sono tipi di dato composti da più elementi anche di diverso tipo.
Un array individua un insieme di elementi **dello stesso tipo** individuabili tramite la loro posizione.
Un record individua un insieme di elementi **di qualsiasi tipo** individuabili tramite il loro nome.
Un array unidimensionale (vettore) è un tipo di dato strutturato caratterizzato da un nome e da un numero di componenti, tutti dello stesso tipo, individuabili tramite un solo indice.
**Esempio**
type Vettore: array [1..5] of integer;
dichiara un vettore di 5 elementi di tipo intero.
Esistono anche gli array multidimensionali, gli elementi di un array multidimensionale si identificano tramite più indici.

# Complessità

Un algoritmo deve essere eseguito rapidamente, il tempo di esecuzione dipende dalla velocità dell'esecutore, dal numero e dal tipo di istruzioni eseguite, il numero di istruzioni da eseguire dipende dai dati da elaborare.
A parità di esecutore, di tipologia di istruzioni e di dati di input per migliorare le prestazioni di un algoritmo bisogna minimizzare le istruzioni da eseguire.
Per esempio, la ricerca completa ha una complessità di tempo che dipende da N/2, mentre la ricerca dicotomica ha una complessità di tempo che dipende da $\log_2 N$, la ricerca dicotomica è più efficiente della ricerca completa, questo significa che a parità di esecutore sarà sempre più veloce.
L'efficienza di un algoritmo dipende da:

- Numero di operazioni logico/aritmetiche effettuate
- Spazio di memoria richiesto dai dati

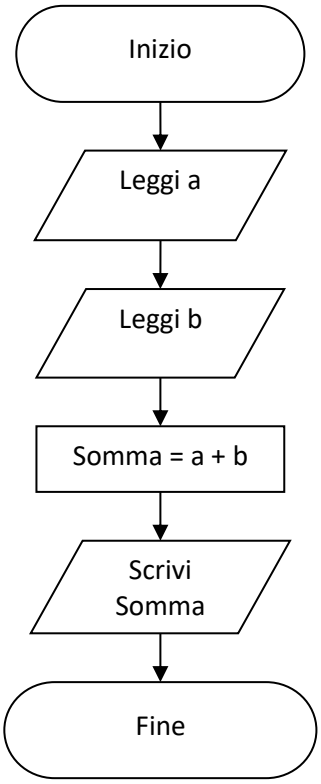
La riduzione del numero di operazioni non permette solo di diminuire il tempo di esecuzione ma anche di ridurre le approssimazioni proprie di alcune operazioni.

# Esempi di scomposizione

## Scomposizione sequenziale

### Somma di due numeri interi.

Leggi a
Leggi b
Somma = a + b
Scrivi somma



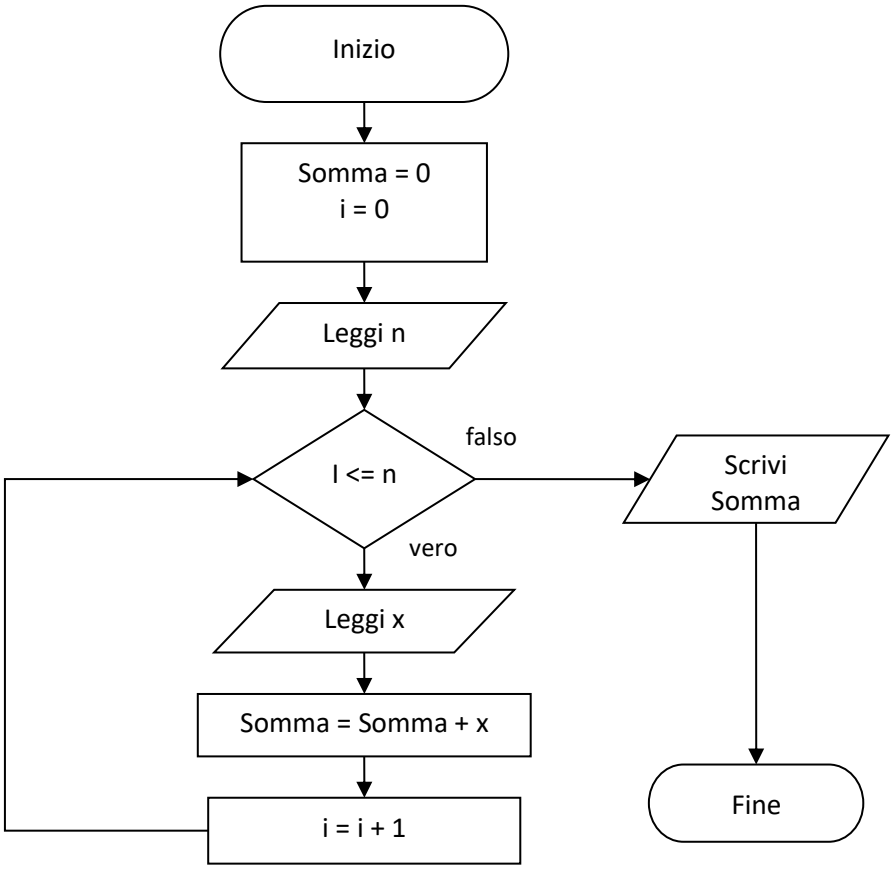

### Somma di n numeri.

**Prima versione**
Leggi $a_1$
Leggi $a_2$
...
Leggi $a_n$
Somma = $a_1 + a_2 +. .+ a_n$
scrivi Somma
**Seconda versione**
Somma = 0
Leggi $a_1$
Leggi $a_2$
...
Leggi $a_n$
Somma = Somma + $a_1$
Somma = Somma + $a_2$
...
Somma = Somma + $a_n$
scrivi Somma

# Scomposizione iterativa

## Somma di n numeri

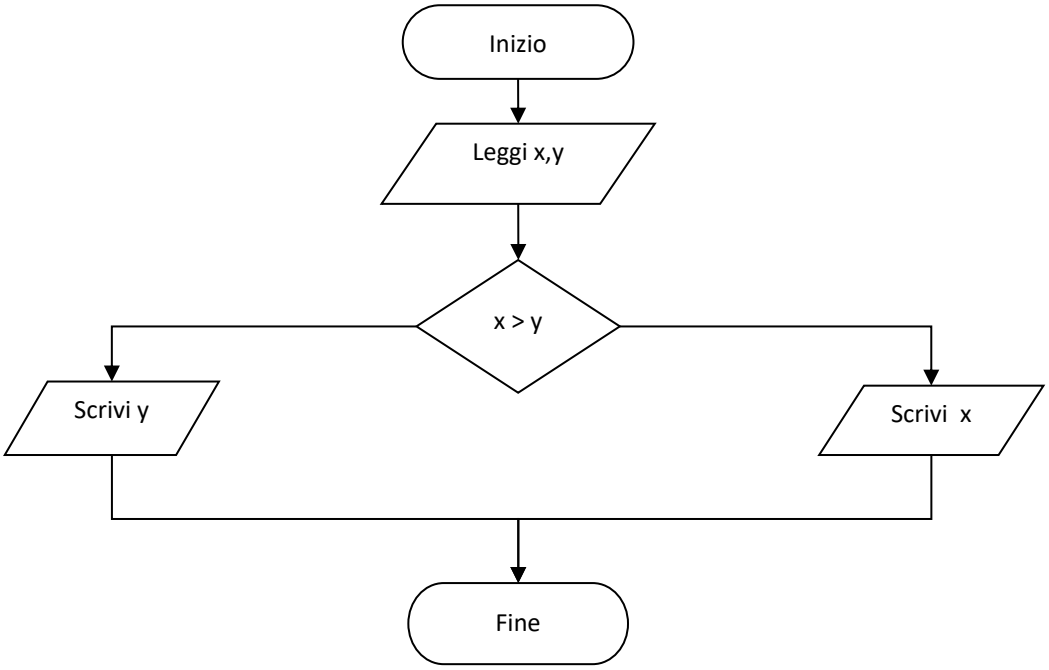
```
i = 1
Somma = 0
Leggi     n
finché    i <= n ripeti
          Leggi     x
          Somma = Somma + x
          i = i+1
fine ciclo
scrivi somma
```




# Scomposizione condizionale

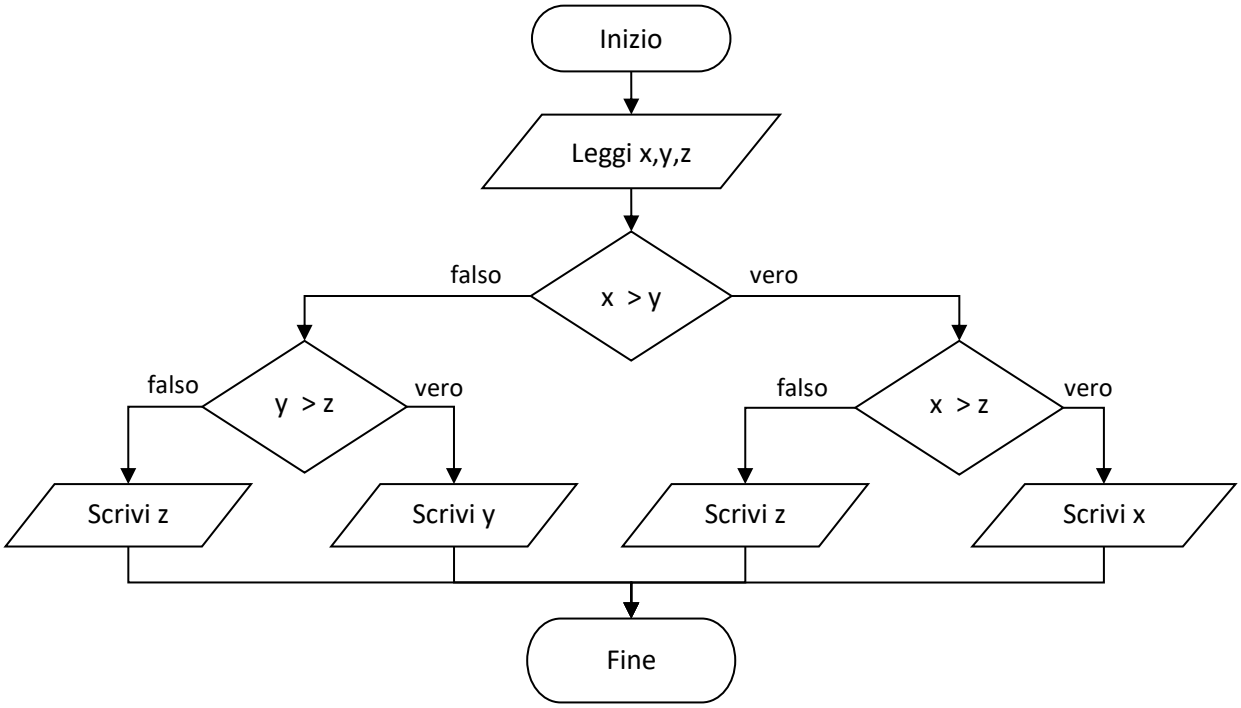
## Determinare il valore massimo tra due numeri x e y

```
Leggi x
Leggi y
se x > y allora
          scrivi x
altrimenti
          scrivi y
fine se
```




## Determinare il valore massimo tra 3 numeri x,y,z.

```
Leggi x,y,z
se x > y  allora
        se x > z allora
                scrivi x
        altrimenti
                scrivi z
        fine se
altrimenti
        se y > z allora
                scrivi y
        altrimenti
                scrivi z
        fine se
fine se
```


Inizio
Leggi x,y,z
x > y
falso
vero
y > z
falso
vero
x > z
falso
vero
Scrivi z
Scrivi y
Scrivi z
Scrivi x
Fine

# Sommario