ALGORITMI

Nell'informatica ci sono essenzialmente due *dati* e *algoritmi*.

Algoritmo : procedimento atto a risolvere un problema, utilizzando dei dati e ottenendo un risultato, rappresentato da un insieme ordinato di passi eseguibili e non ambigui.

ALGORITMO PER LA SOMMA DI DUE NUMERI

In generale, un algoritmo riceve un insieme di valori (dati) in input e ne genera uno in output (chiamato soluzione).
In questo caso dati di partenza (input) sono i dati di ingresso *variabili* a e b, il risultato cercato (output) è il numero c = a + b

Es. impostiamo:

a=2

b=2n

c=a+b

Le istruzioni che il calcolatore dovrà seguire sono:

Prendi il primo numero a=2

Prendi il secondo b=2n

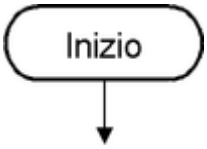
Fai la somma c=a+b

PREMESSA

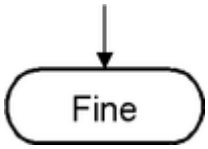
Per scrivere la sequenza di passi, ovvero l'algoritmo, che il calcolatore dovrà seguire per arrivare alla soluzione del problema in questione utilizziamo uno *schema a blocchi o diagramma di flusso*
I diagrammi di flusso permettono di descrivere in modo grafico, sottoforma di uno schema formato da blocchi, le azioni che costituiscono un algoritmo e il loro flusso di esecuzione.
Ogni blocco contiene un' azione elementare. I singoli diagrammi devono essere uniti tramite i *connettori.*
L'esecuzione delle istruzioni deve essere fatta *sequenzialmente*, ovvero seguendo i connettori.

Esistono 5 tipi di blocchi elementari che vengono utilizzati a seconda dellì'istruzione o dei dati che contengono:

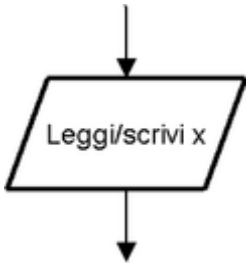
blocco iniziale



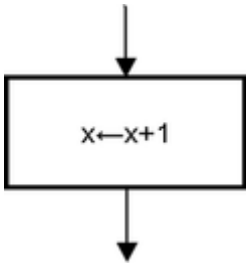
blocco finale



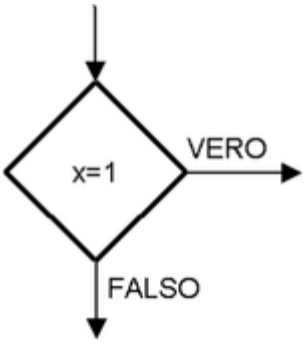
blocco di I/O



blocco di Elaborazione



blocco di controllo



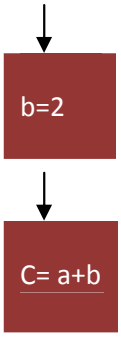
Per quanto riguarda il nostro “algoritmo somma” ,considerando i dati suddetti, le operazioni che dovremo svolgere sono dette di assegnazione.

Istruzione di *assegnazione* : “ ← ”

• Variabile ← Espressione;

– Es.: a ← 2; b ← 2n; c ← a + b;

a=2



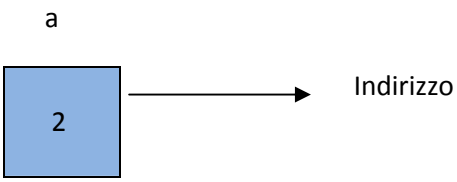
La lettera **a** rappresenta una variabile, quindi può assumere qualsiasi valore numerico e può cambiare nel tempo, in questo caso assegniamo alla lettera **a** il valore 2.
La variabile è caratterizzata da un TIPO: in questo caso, avendole assegnato il valore due, un tipo numerico.
N.B l'operazione di assegnazione differisce dall'operazione logica di confronto.

Per la variabile **b** l'istruzione è la stessa cambierà solo il valore assegnato.

La terza istruzione è **c=a+b** che è sempre un operazione di assegnazione ma preceduta dal calcolo del valore **a+b** ,il cui risultato sarà appunto il valore relativo a **c**.

A **c=a+b** corrisponde invece un'altra istruzione per cui il calcolatore dovrà conoscere l'indirizzo della cella dove sarà posto il valore della prima e della seconda variabile, che rappresentano gli operandi, e poi quella della terza relativo alla loro somma.

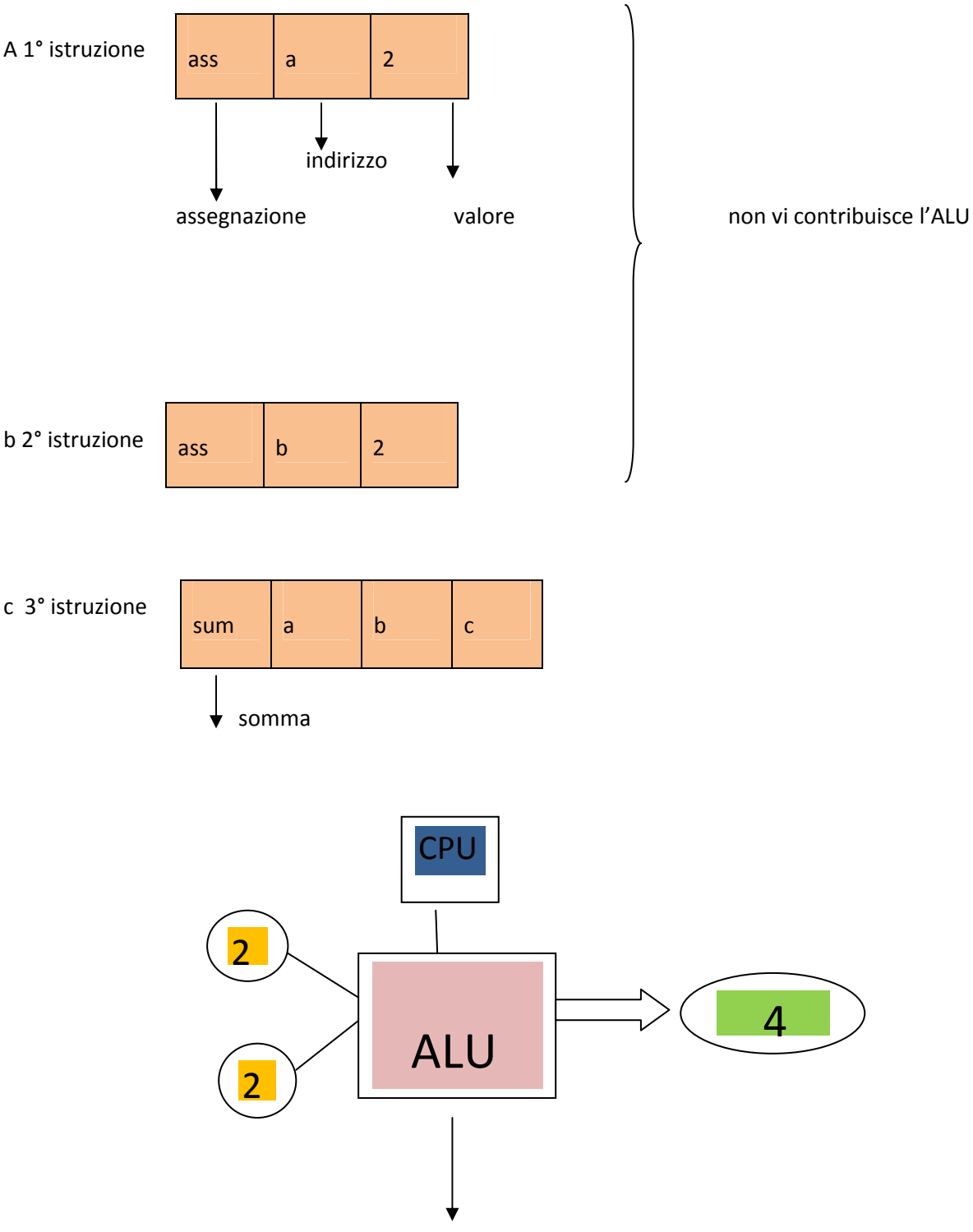
Dal punto di vista dell' architettura del computer la lettera **a** è una cella di memoria vuota.Ogni cella di memoria ha un suo indirizzo quindi il computer dovrà prendere l'indirizzo di **a** ed inserirci il valore che gli è stato assegnato.
Dunque il computer per svolgere l'operazione deve avere una serie di istruzioni (in questo caso un *assegnazione*) che legge dal PC, l'indirizzo della cella dove andare a scrivere ed infine il valore da scrivere nella stessa cella. Quindi dovrà accedere alla memoria per scrivere il valore due all'interno della cella.



Per quanto riguarda lettera **b**, come già detto, l'istruzione è la stessa cambierà solo l'indirizzo ed il valore corrispondente.

Per **c=a+b** la CPU andrà a leggere il valore delle variabili dalle corrispondenti celle della memoria così la ALU legge il valore di **a** trova due e lo mette nel secondo registro dove si svolge l'operazione, si ripete la stessa operazione per il valore **b**, e poi si occupa dell'operazione somma che non viene però svolta in quanto dal

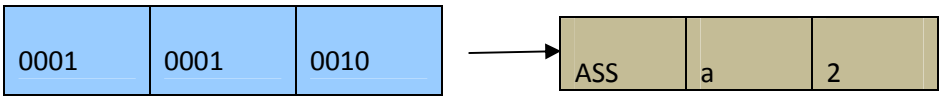
punto di vista del computer sarà secondo codice binario e quindi avremo una tabella caratterizzata da una corrispondenza tra stringhe di bit e operazioni.



SOMMA : istruzione data dalla CPU

Dunque anche se da un punto di vista astratto descriviamo l'algoritmo con un diagramma di flusso questo non è in realtà comprensibile dal calcolatore il cui linguaggio è basato su un alfabeto detto binario i cui simboli sono i bit. Questi simboli (1 e 0) sono organizzati in "parole" che a loro volta costituiscono "frasi" dette istruzioni e rappresentate da stringhe di bit. Ognuna di esse ordina al processore di eseguire un'azione elementare afferente allo stato interno del computer come, riferendoci ancora al nostro esempio, la lettura di una locazione di memoria oppure il calcolo della somma dei valori contenuti nei due registri dell'ALU.
Questo appena descritto è detto *linguaggio macchina* per distinguerlo dal *linguaggio naturale* con cui ci noi esprimiamo l'algoritmo nel diagramma di flusso.

LINGUAGGIO MACCHINA :



E' logico supporre che per poter arrivare da un linguaggio all'altro ci sia bisogno di un passaggio intermedio di traduzione degli algoritmi scritti nel linguaggio naturale in bit.
Infatti, affinché una macchina riesca a comprendere ed eseguire i passi specificati da un algoritmo, quest'ultimo deve essere prima codificato in un opportuno *programma*
Dunque dopo aver impostato lo schema a blocchi, necessario per avere una prima inquadratura degli algoritmi, questi dovranno essere tradotti in una sequenza di istruzioni che vengono scritti in l*inguaggi di programmazione* i più conosciuti dei quali sono il pascal, il basic, il "c"

In generale un *linguaggio di programmazione* è un linguaggio formale, dotato di un lessico, una sintassi e una semantica ben definite, cosa che rende possibile scrivere i programmi partendo dagli schemi a blocchi. Questa traduzione dal linguaggio naturale al linguaggio macchina viene effettuata da un programma detto *compilatore*: il risultato è un file binario 'eseguibile' ovvero stringhe di 1 e 0.

Dunque passaggi che si devono seguire per scrivere un algoritmo e renderlo accessibile al calcolatore sono:

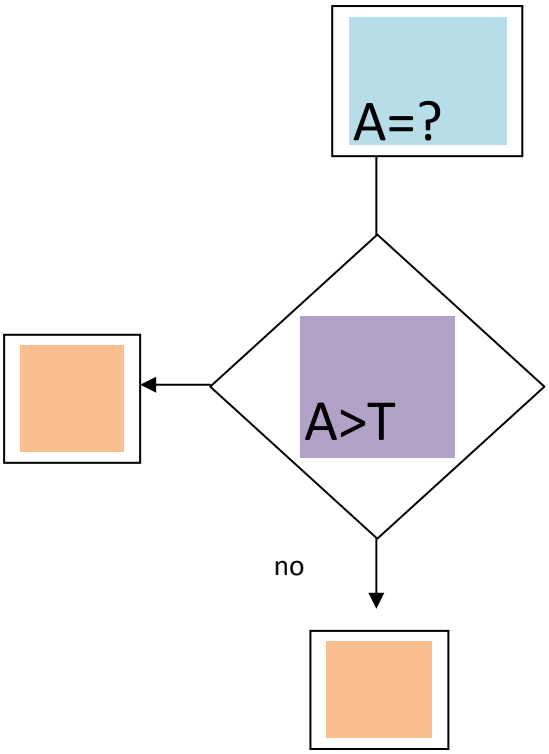
1. Scrivere l'algoritmo in linguaggio naturale, a blocchi
2. Tradurlo in linguaggio di testo
3. Traduzione in linguaggio macchina da parte del compilatore
4. Memorizzazione nella memoria del calcolatore (l'algoritmo è pronto per essere eseguito)

Algoritmi decisionali

Oltre quella di assegnazione un altro tipo di operazione fondamentale in informatica è quella che permette di prendere delle decisioni, ovvero data una determinata variabile **a** input, a seconda del valore che questa

assume, potrà essere vera o falsa e,in relazione a questo risultato, il calcolatore svolgerà una determinata operazione o il ciclo potrò arrestarsi.

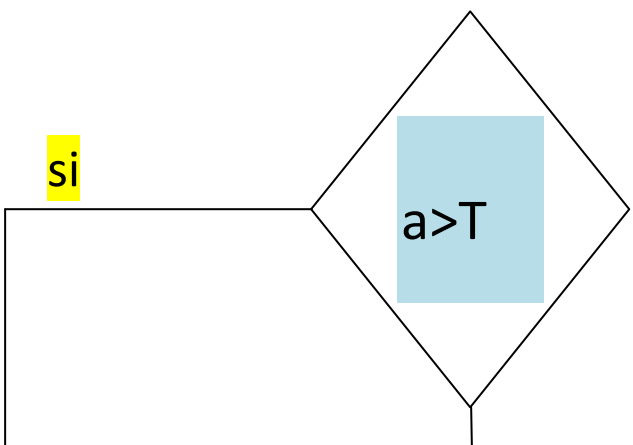
Per costruire il diagramma di flusso di questo tipo di algoritmo utilizzeremo il blo*cco di controllo* con una sola freccia entrante e due frecce uscenti corrispondenti ai valori vero e falso.



Dato un valore assegnato alla variabile **a** la decisione consiste nel confronto tra questa variabile e un valore di riferimento: se **a** soddisfa la condizione di confronto (ad esempio a>t) svolgeremo un operazione, in caso contrario un'altra (si/no nello schema).

La sintassi di questo diagramma in linguaggio pascal è scritta:

if(a>t)then….else



…else

no

than…

quindi questo blocco con le sue dilatazioni viene tradotto come

“se questa cosa è così allora fai questa cosa altrimenti fai un'altra cosa”

Nel nostro esempio a seconda del valore di **a** l’istruzione può essere eseguita oppure no,
da questo si può dedurre che è importante che il linguaggio di programmazione preveda un istruzione di “salto” ovvero dia la possibilità saltare un blocco di istruzioni.

istruzioni di salto

particolari istruzioni che consentono di cambiare il flusso dell’esecuzione, sia a fronte di particolari *eventi* verificatisi, sia incondizionatamente

Per esempio nel linguaggio basic è presente un istruzione detta *“go to”* basata sul fatto che in questo linguaggio di programmazione le istruzioni sono numerate.

10	A=1

Linguaggio basic: Es. go to

20	B=1
30	A+b=c
40	Go to 10

È in realtà un istruzione importante a livello del linguaggio macchina, dove consiste semplicemente nel saltare delle istruzioni aggiornando il contenuto del PC, necessaria perché altrimenti non potrei prendere delle decisioni che consistono in deviazioni dal flusso principale.

Capita che una certa istruzione debba essere svolta n volte

Supponiamo di dover stampare tutti i numeri.

Avremo una serie di istruzioni ripetitive ovvero l'assegnazione di **a** e la stampa di **a**

Potremo porre un diagramma in cui abbiamo due blocchi corrispondenti all'informazione di assegnazione alle variabili **c** ed **a** ed uno a quella di stampa.
Porremo poi in un ulteriore blocco un informazione decisionale data la quale se **a** è minore di 10 avremo l'incremento di **c** ovvero l'assegnazione **c**=**c**+1 tramite la quale **c** cambia valore e il ciclo riprende.

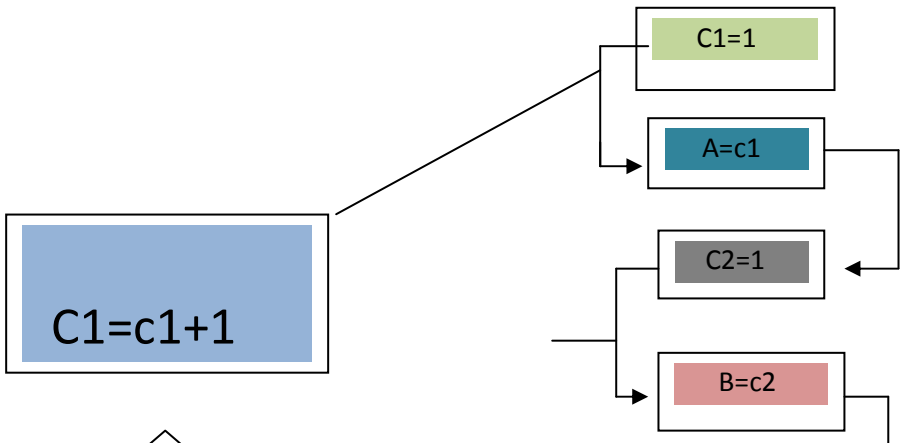
In questo modo potremo svolgere delle operazioni ripetitive senza dover scrivere n volte la stessa informazione.

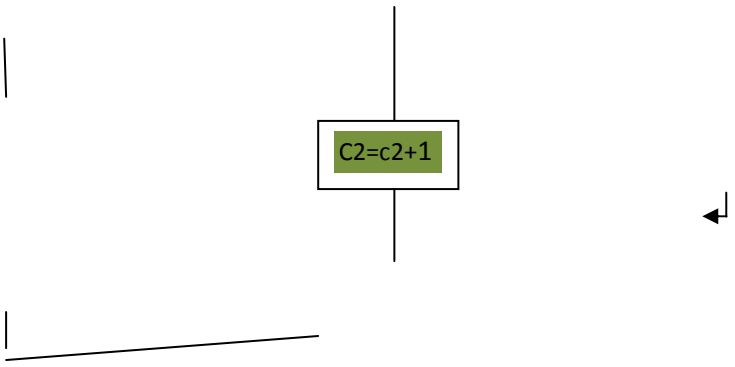
Es. Tabellina pitagorica.

Avendo delle righe e delle colonne non basta una sola variabile ma ne servono due.
Faremo dunque un primo contatore **c**□=1 poi un secondo **c**□=1 per poi porre **a**=**c**□**; b**= **c**□**; d**=**a**×**b**

Supponiamo di mantenere fisso **a** e far variare **b** ovvero:

b>10 *if* yes *then* → incremento il contatore (**c**□=2) e quindi assegno un altro valore a **b**





Quindi all'inizio **a** e **b** saranno uguali ad 1, al passaggio successivo **a** sarà sempre uguale ad 1 ma a **b** verrà assegnato il valore 2.
In questo modo al termine di 10 passaggi avremo la prima riga della tabellina.
Nel momento in cui **b**>10 si uscirà dal ciclo andando a incrementare **a**.
Troveremo così l'assegnazione **c**□= **c**□+1 che determinerà il cambiamento della variabile **a** mentre **b** rimarrà fissa al contrario dei primi passaggi.
Anche l'incremento di **a**, come quello di **b**, sarà bloccato da un informazione decisionale che al valore di **a**>0 farà corrispondere la conclusione del ciclo e quindi il blocco fine.

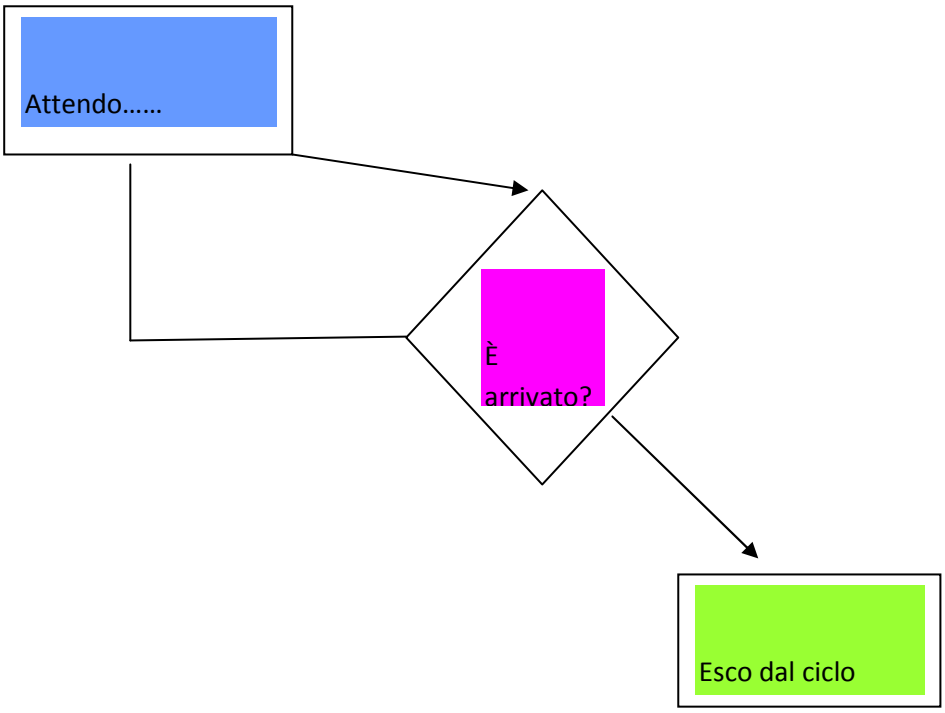
Come possiamo notare queste operazioni cicliche sono caratterizzate dalla presenza di un *contatore*.
In un ciclo semplice, con una sola variabile, possiamo trovare anche un solo contatore, ma all'aumentare delle variabili dovranno essere inseriti più contatori.
I contatori vengono incrementati da un *test* che decide quando uscire dal ciclo,

Ciclo for e ciclo while do

Linguaggio di programmazione "c": for(a=1; a<10; a++) □ prunt a □

Per **a** che va da uno a 10 fintanto che **a** è minore di 10 ad ogni passo incrementare **a** di uno e poi svolgere l'operazione.

Questo è un ciclo detto *"for"* che ha un termine preciso data la presenza di un contatore e quindi il limite di a<10.
Esistono anche dei cicli di cui non è noto il termine come nel caso di quello di input della tastiera di una volta caratterizzato da un test ripetitivo del tipo "è arrivato il carattere?" a cui al valore si era associato l'uscita dal ciclo ed al valore no la modalità di attesa a cui si ripeteva lo stesso test.



Quindi mentre nel ciclo *for* le operazioni iniziali, cioè inizializzazione e termine, sono regolate dal contatore e il termine è unitario e predefinito, in questi tipi di cicli detti "*while do*" l'istruzione non viene ripetuta per un numero predeterminato di volte, ma viene eseguita se si verificano certe condizioni e ripetuta fino a che tali condizioni sono verificate;dunque mentre nel ciclo for il numero di interazioni viene specificato in forma esplicita e quindi impostiamo per quante volte deve essere ripetuta, nel ciclo while do non è noto il numero di interazioni che avremo.

FUNZIONE

Una funzione è un'operazione che partendo da certe variabili restituisce un risultato.

Nel linguaggio di programmazione "c" tipicamente si usa una *parola chiave* per descrivere tale operazione, ove le parole chiave sono i simboli come *if* e *then* così chiamati per distinguerle dalle variabili.
La parola chiave che introduce una funzione è *function*.

La funzione è caratterizzata da una struttura di questo tipo:

A=somma (b,d)

Dopo l'uguale troviamo un nome che serve a richiamare il tipo di funzione, ma che non deve necessariamente richiamarla dal punto di vista logico, ad esempio una funzione chiamata "somma" non per forza corrisponderà all'operazione somma.
Le lettere tra parentesi indicano gli argomenti o parametri della funzione.

Es. funzione somma

Function a=somma (b,c) □a=a+b□

Tra parentesi graffe troveremo l'operazione che svolge la funzione.
Al termine solitamente è presente un istruzione di detta *return A;* tramite la quale la funzione restituisce alla variabile **a** il valore che gli è stato assegnato tramite l'operazione.
Possiamo avere più valori di ingresso mentre solitamente solo uno in uscita nel linguaggio di programmazione "c".
L'utilità della funzione è quello di raggruppare un certo gruppo di istruzioni che svolgono sempre la stessa operazione e che è possibile richiamare; Per descrivere lo stesso raggruppamento di operazioni in un algoritmo generale avremmo dovuto infatti impostare le varie variabili, assegnare tutti i relativi valori ed impostare uno schema decisionale, operazione ovviamente molto più lunga.

Il concetto base delle funzioni è il riutilizzo delle strutture

es. mettere in ordine alfabetico.
Applicazioni: elenco studenti, elenco telefonico tutti i programmi che hanno al loro interno l'ordinamento della lista

Quindi le funzioni sono la base di algoritmi più complessi che si formano appunto tramite la loro unione.

PROCEDURA

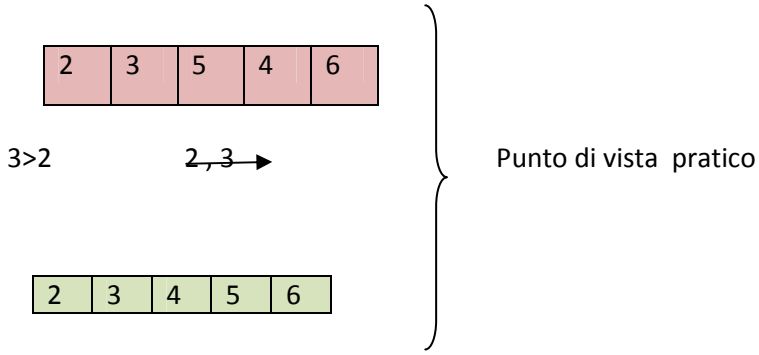
Funzione che non restituisce il valore in uscita, pacchetti predefiniti di operazioni.

ALGORITMO TABELLINA PITAGORICA LINGUAGGIO C

```
For (a=1;a<10;a++)
For(b=1;b<10;b++) □prin t (a*b);□
```

Il secondo ciclo for è contenuto all'interno del primo. Questi due cicli vengono perciò detti *innestati* (uno dentro l'altro) e servono a stampare la tabellina pitagorica.

ALGORITMO PER ORDINARE UNA LISTA



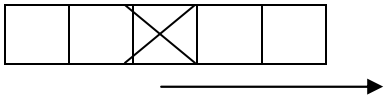
3,2,5,4,6 mettere in ordine crescente

3,2 → 3>2→2,3 (scambio)

3,4→3<4→3,4 (sono in ordine? Si allora non scambio)

ARRAY VETTORE

Ā



array

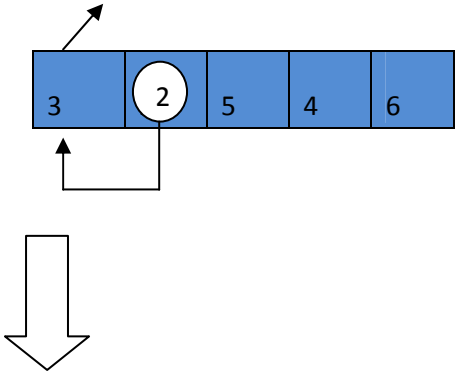
Con una sola variabile possiamo indicare un'intera cella di memoria.
Per prendere un elemento all'interno di questa cella **a** detto arrey o vettore si utilizza una particolare sintassi ovvero *a[3]*

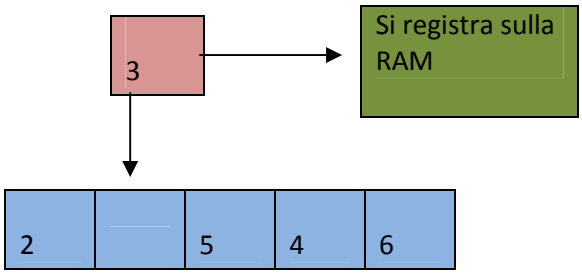
If A[n]<a[n+1+1] è vero THEN niente
non vero ELSE swap (scambio)

SWAP bubble sort (ordinamento scambio a bolle)

Lo scambio si fa memorizzando in una cella il valore che devi scambiare e poi ponendo per fare scambi due alla volta:

3





D=a[n]

a[n]=a[n+1]

A[n+1]=d

D = *variabile temporania* a cui si assegna il valore che deve essere momentaneamente memorizzato.

Nel momento in cui vi è uno scambio poniamo la *flag*=vero; il ciclo termina solo quando *flag=falso* ovvero quando non ci sono avvenuti scambi e l'operazione è stata completata.

La nostra lista di esempio era suscettibile ad essere ordinata dopo il primo passaggio ma il processore deve avere un modo per rendersi conto che la lista è ordinata quando va a rifare la scansione dei dati ed è appunto nel momento in cui non opera scambi che può essere certo che la lista è ordinata.

Questa operazione deve essere inclusa in un ciclo *or* che innanzi tutto scandisce tutta la lista, incrementando ogni volta il valore di **n** fino al termine della lista, poi è necessario un ciclo di tipo *while* in modo tale che fin quando avvengono scambi si debbano eseguire gli *swap* e solo quando cessano ciclo è terminato.
Vengono utilizzati cme visto anche le *flag* bandierine che si impostano vere o false a seconda che ci siano stati o meno scambi.
Dunque la cpu prima della fine del ciclotesta il flag.

While flag flag=false for (n=1;n<L,n++)

If(a[n]<a[n-1])

ALGORITMO PER RICERCHE GOOGLE

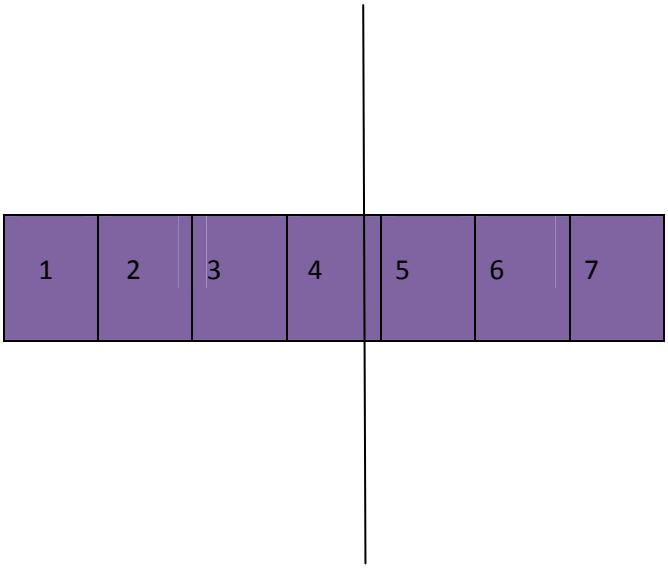
Per prima cosa dovremo porre un algoritmo per ordinare una lista quindi poniamo function a=ordina (a)

Dove **a** in entrata è una lista non ordinata mentre la stessa variabile in uscita sarà una lista ordinata degli stessi elementi.

Es 1 2 3 4 5 6 7

Il mio obiettivo è quello di trovare il numero 1 all'interno di questa lista.

Procediamo dividendo la lista a metà.



1. CERCA 1= si divide a metà cioè a 4
2. Poi 4 è > di 1? Si allora si divide ancora a metà cioè a 2
3. 2 è > di 1? Si trovato il valore 1

Quindi una volta divisa a metà la lista se al margine della parte di lista in cui ho diviso l'iniziale trovo un numero di cui il nostro è minore:

4567

Il numero da noi cercato dovrà trovarsi prima di esso ovvero alla sua sinistra dato che la lista è crescente

Nel caso invece il nostro numero sia maggiore di esso lo troveremo alla sua destra.

Dunque opero per *successive divisioni* della lista

Traducendo nel *linguaggio di programmazione* avremo una lista ordinata di elementi caratterizzati da un indice iniziale 1 e un indice finale 7.

Prendo l'elemento di centro 7/2.

Cerco C.

Se l'elemento al centro è minore di C allora C sta a destra dell'elemento di 7/2

Scarto la parte a sinistra dell'elemento di centro 7/2.

RIDEFINISCO L'INDICE INIZIALE E L'INDICE FINALE ponendoli ora come 7/2 e 7 (avendo considerato la seconda parte della lista dove si trova C)

IF (a [L/2]<C)

Secondo passaggio (incremento)

Si cambia l'indice finale e iniziale con i nuovi indici finale e iniziale nella funzione ponendo:

INDICE INIZIALE: Finale meno iniziale diviso due più parte iniiziale;
INDICE FINALE: Finale mezzi (sulla lista di destra)

Sintassi ricerca di un nome all'interno di una lista

A=ordina(a)

N=cerca (c)

TIPIZZAZIONE VARIABILI

Numeri interi

Int a;→ la variabile **a** è di tipo intero (numero intero) la cella di memoria dove si collocherà sarà caratterizzata dal fatto che contiene numeri interi che potranno essere insigned oppure signed

Insigned int a;→senza segno

signed int b; →con segno occupando un bit per il segno

A seconda dell'ordine di grandezza dei numeri che vogliamo utilizzare

signed int : 1byte

Insigne int : 2byte

Long it : 4byte

Numeri con la virgola

Double: numero con la virgola con mantissa e esponente

Double b :

l double b; -→ long double

È importante che questi tipi tra di loro siano compatibili perché ad esempio un tipo int può contenere fino a 16.000 numeri mentre un chard fino a quindi si può avere incompatibilità, mentre double e int sono compatibili, ovvero il risultato di un'operazione tra questi due tipi può andare in uno o nell'altro perchè rappresentano entrambi un numero molto alto di valori numerici.

Vettori

Int a [10];→ vettore di 10 elementi

Double a [10];→vettore in cui gli elementi sono numeri con la virgola

STRUTTURA GENERALE DI UN PROGRAMMA

Dichiarazione delle variabili

Dichiarazioni di funzione function a=somma (a,b)

MAIN programma principale che utilizza le variabili e le funzioni per concepire l'algoritmo.

A=ingesso dati

A=ordina (a)

Out put (visualizza i risultati)

Return

Questo programma principale viene convertito dal processore in una serie di stringhe di 1 e zero e posto in una memoria.

SISTEMI OPERATIVI
SOFTWEAR CHE GESTISCONO LA MACCHINA

OS. operative sistem

Il primo sistema operativo si chiamava MS-DOS ( )

Il sistema operativo gestisce le memorie di massa e quelle virtuali (necessarie perchè la ram può memorizzare solo una certa quantità di dati e nel momento in cui è satura farà uno swap liberando uno spazio per mettere dentro i dati che servono in quel momento ovvero dell'applicazione corrente , quindi le applicazioni che in quel momento sono oscurate possono mettere i dati momentaneamente sul disco)

KERNEL nucleo di un sistema operativo che ha la funzione di fornire ai processi in esecuzione sull'elaboratore un accesso sicuro e controllato all'hardware. Dato che possono esserne eseguiti simultaneamente più di uno, il kernel ha anche la responsabilità di assegnare una porzione di tempo-macchina e di accesso all'hardware a ciascun programma .Rappresenta l'applicazione ovvero file exe (nome.exe) in linguaggio macchina.

La CPU esegue solo ciò che è scritto nel PC quindi quando carico in memoria il programma scritto sul file lo carico in una determinata zona della memoria poi devo prendere questo indirizzo di memoria e metterlo nel PC per consentire alla cpu di eseguirlo. Il Kernel ha il compito di prendere il programma metterlo nelle celle libere per poi caricare la cpu con questo programma.Il programma al suo interno può fare riferimento ha delle variabili che si traducono in delle locazioni ovvero delle celle di memoria.
Tipicamente algoritmi e dati non sono nella stessa regione quindi il programma deve sapere dove sono le variabili di cui al suo interno avrà gli indirizzi di memoria che però non sono assoluti perchè le variabili si pongono dove trovano gli spazi liberi nel momento in cui l'indirizzo a cui in realtà corrispondono è occupato.

Si dice che gli indirizzi delle variabili sono RELATIVI infatti sono posti in maniera tale che tra di loro le variabili siano contigue ma appunto come detto possono variare per la necessità di liberare la memoria. Dunque per trovarle abbiamo bisogno di un riferimento, che viene fornito dal kernel. dell'indirizzo attualmente utilizzato dalle variabili *ALLOCAZIONE DINAMICA* appunto caratterizzata da riferimenti non assoluti ma relitivi dove il primo indirizzo viene dato dal kernel e le postazioni delle seguenti variabili sono relative appunto a questo primo indirizzo essendo collocate in maniera contigua.

Il programma può utilizzare delle SUB RUTIN (qualcosa di frequente ma che non fa parte del programma principale)dunque possiamo dire che il sistema operativo è un po' come il meccanismo delle intrerruzioni il kernel sposta l'attenzione dal programma principale alle sub rutin

*Nome.DLL (DYNAMIC LINKED LIBRERIA-libreria caricate dinamicamente)*il sist operativo ha a disposizione questi DLL che vengono caricate quando necessarie dal kernel

MULTITASKIN (molti compiti in contemporanea): Il processore può svolgere un'operazione alla volta ma il kernel è in grado di saltare da un operazione all'altra quindi il processore in realtà non fa mai più operazioni contemporaneamente ma le alterna-

- LUNIX o LINUX (multitaskin) sistemi operati iniziali più diffusi.
- MS.DOS(singol taskin).
- WINDOWS (multitaskin).

Ogni applicazione può durare un totale di tempo prefissato trascorso il quale si passa ad un'altra applicazione.Il kerner gestisce questa modalità di esecuzione sguizzando ovvero cambiando ogni tot di tempo l'operazione della macchina.Questo tipo di multitaskin è detto PREEMPTIVE completo ed è caratterizzato dal fatto che è il kernel a decidere quanto tempo si ha la possibilità di utilizzare l'applicazione ovvero il controllo è del kernel.