LABORATORIO DI INFORMATICA

Prof.ssa Patrizia Tarantino

1. Modulo 1 - Problemi ed algoritmi
2. Modulo 2 – Prompt dei comandi
3. Modulo 3 - Linguaggi di programmazione C e C++

INDICE

MODULO 1 - PROBLEMI ED ALGORITMI

Concetto di PROBLEMA

Trovarsi in una situazione che ci pone delle domande, alle quali dobbiamo dare delle risposte, significa avere un **problema** da risolvere.

Un problema è caratterizzato da:

- **Dati iniziali:** ossia ciò che ci è noto e che indichiamo col termine **input**
- **Dati finali o Risultati:** cioè gli elementi incogniti che si devono determinare e che indicheremo con **output**
- **Calcoli o Elaborazioni:** sono le azioni (istruzioni) che consistono nella **soluzione,** ovvero che combinate porteranno al raggiungimento dei **risultati**.

CLASSE di PROBLEMI

Nella **CLASSE DI PROBLEMI**, i dati, (o almeno uno) sono forniti come parametri generici.

Esempio: Calcolare il perimetro del quadrato di lato L

ISTANZA di PROBLEMA

Nella **ISTANZA DI PROBLEMI**, tutti i dati, sono forniti con valore specifico .

Esempio: Calcolare il perimetro del quadrato di lato 5.

Definizione di ALGORITMO

Informalmente, un **algoritmo** è una procedura ben definita che serve per risolvere un dato problema

E' una sequenza di passi che prende dei valori come *input* e produce altri valori come *output*

Più precisamente, un algoritmo è descrivibile con un **numero finito di operazioni che conducono al risultato**

IL PROGRAMMA

In ambito informatico, l'algoritmo viene descritto tramite un **programma**, ossia un testo scritto in uno specifico linguaggio detto **linguaggio di programmazione** per essere eseguito da calcolatori elettronici

Ogni linguaggio di programmazione ha una sua **sintassi** ed una propria **rappresentazioni dei dati**, e utilizza le proprie **operazioni di manipolazione dei dati**

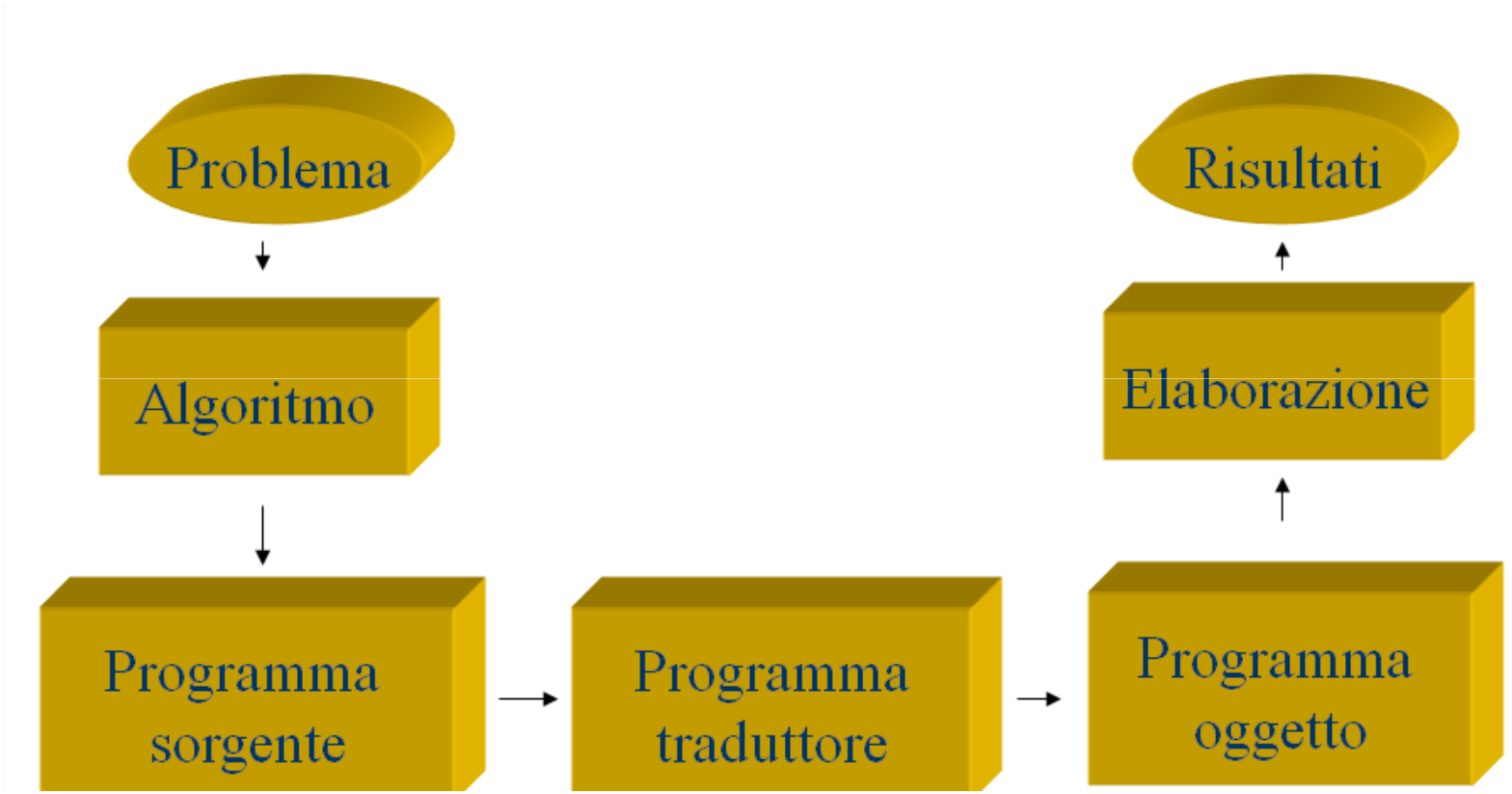
Invece:

Le proprietà degli algoritmi sono talmente **fondamentali, generali** e **robuste**, da essere **indipendenti** dalle caratteristiche di specifici linguaggi di programmazione o di particolari calcolatori elettronici

Quindi:

PER RISOLVERE UN PROBLEMA E' ASSOLUTAMENTE NECESSARIO TROVARE UN PROCEDIMENTO RISOLUTIVO DESCRIVIBILE CON UN ALGORITMO, E SOLO DOPO SI PUO' PASSARE A SCRIVERE IL PROGRAMMA

DAL PROBLEMA AL PROGRAMMA



PROGRAMMA = ISTRUZIONI + DATI

Un programma può essere visto come un manipolatore di dati:

a fronte di dati in ingresso che descrivono il problema producono dati in uscita come risultato del problema

Le operazioni sui dati sono le **istruzioni.**
I dati sono gli **oggetti** su cui vengono eseguite le **istruzioni.**

Tutti i linguaggi di programmazione prevedono un insieme di
dati e istruzioni
che il programmatore può utilizzare nella realizzazione dei propri programmi.

I DATI

I dati possono essere distinti in base a :

- **Utilizzo** all'interno del processo
  - INPUT
  - OUTPUT
  - LAVORO
- **Formato**
  - NUMERICO
    - INTERO
    - DECIMALE
  - ALFANUMERICO
- **Valore** assunto durante il processo
  - VARIABILE
  - COSTANTE

Questi aspetti dei dati devono essere definiti già nell'algoritmo, in modo da poterli poi "tradurre" con la sintassi propria del linguaggio di programmazione

LE ISTRUZIONI

Anche le istruzioni devono essere scritte rispettando la **sintassi** propria **del linguaggio di programmazione**

Ogni linguaggio di programmazione ha le proprie istruzioni, ma tutti i linguaggi devono possedere alcune istruzioni per compiere le azioni fondamentali

- **Acquisire** i dati su cui attivare il processo di elaborazione → **input**
- **Comunicare** all'esterno i risultati ottenuti → **output**
- **Azioni di tipo aritmetico** → **assegnazione**
  - assegnare un valore ad una variabile
  - utilizzare gli operatori aritmetici (+, -, /, *, %)
- **Azioni di tipo logico** → **confronto**
  - Utilizzare gli operatori di relazione(=, <, >, <=, >=, <>) per confrontare il contenuto di due variabili
  - Utilizzare gli operatori logici (and, or, not)

Programma sorgente

L'algoritmo risolutivo viene trasformato in un programma attraverso un linguaggio di programmazione che può contenere:

- **Istruzioni di dichiarazione**: Descrivono dati e variabili utilizzati dal programma, definendone tipo e struttura
- **Istruzioni di assegnazione**: Consentono di assegnare alla variabile un valore dello stesso tipo della variabile
- **Istruzioni di controllo**: Sono istruzioni che richiedono salti di sequenza nell'esecuzione del programma
- **Istruzioni di input e output**: Richiedono l'ingresso o l'uscita di un'informazione da una periferica alla memoria centrale e viceversa

Gerarchie dei Linguaggi Di Programmazione

L' hardware del computer ha un suo linguaggio di programmazione detto

linguaggio macchina

Il linguaggio macchina manipola direttamente le sequenze di bit fornite dall' hardware, utilizzando le operazioni primitive dell' hardware stesso (operazioni aritmetiche).

Programmare in linguaggio macchina è **faticoso, costoso e comporta degli errori**.

Fin dagli anni 50 sono stati sviluppati linguaggi più evoluti, ciascuno dei quali progettato su un linguaggio più rudimentale.

Linguaggi di programmazione a basso livello

- **Linguaggio macchina**

Le operazioni disponibili sono quelle direttamente fornite dall'hardware; ogni operazione è codificata da una sequenza di bit; ogni dato è indicato dall'indirizzo binario della parola di memoria in cui è memorizzato. Ogni indirizzo è espresso in modo assoluto (rispetto a tutta la memoria disponibile). Un programma è una sequenza di bit, che viene direttamente interpretata dall' hardware.

- **Linguaggio assembler** (o assemblativo) Le operazioni sono quelle direttamente fornite dall' hardware, ma sono indicate da nomi convenzionali; i dati sono indicati da nomi, che corrispondono a indirizzi.

Gli indirizzi sono espressi in modo relativo rispetto all'inizio del programma, permettendo così modifiche più semplici.

Il programma, per essere eseguito, viene tradotto in linguaggio macchina da un programma traduttore detto **assemblatore**.

Linguaggi di programmazione ad alto livello

Linguaggi procedurali

Le operazioni disponibili sono ampie e non legate a quelle fornite dall' hardware. I dati sono indicati in modo totalmente indipendente dalla loro memorizzazione. Si tratta di **linguaggi progettati affinchè la scrittura dei programmi sia semplice, e dunque, di facile comprensione e verifica**. Per essere eseguito, un programma deve essere tradotto in linguaggio macchina da un **programma traduttore** detto **compilatore** , oppure deve essere interpretato (cioè eseguito da un altro programma, l' **interprete**).

Linguaggi procedurali di rilievo sono as esempio FORTRAN, COBOL, BASIC, Pascal, C. Tra i linguaggi procedurali alcuni sono detti orientati agli oggetti; tra questi ricordiamo C++ e Java.

DIFFERENZE TRA COMPILATORE E INTERPRETE

Compilatore

- Il programma compilatore analizza e traduce nella sua interezza il programma sorgente.
- Il risultato della compilazione è un programma oggetto.
- Solo dopo la traduzione dell'intero programma è possibile eseguirlo

Interprete

- Il programma interprete analizza e traduce istruzione per istruzione in linguaggio macchina.
- Non appena una istruzione è interpretata può essere eseguita.

- La compilazione porta ad applicativi più veloci rispetto all'interpretazione a discapito del tempo di sviluppo.

- L'interprete consente tempi di sviluppo minori a discapito dell'efficienza nell'esecuzione.

Rappresentazioni grafiche e formalizzate di un algoritmo

La descrizione delle fasi esecutive del problema può avvenire mediante la formalizzazione dei passi elementari da effettuare che può essere realizzata con:

Diagramma a blocchi

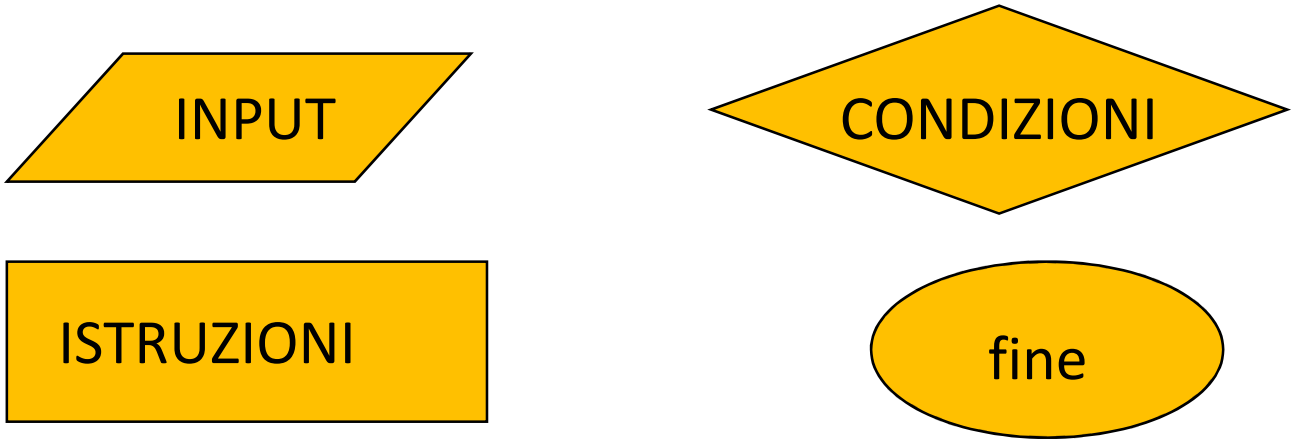
o flow-chart

e/o

Pseudocodifica

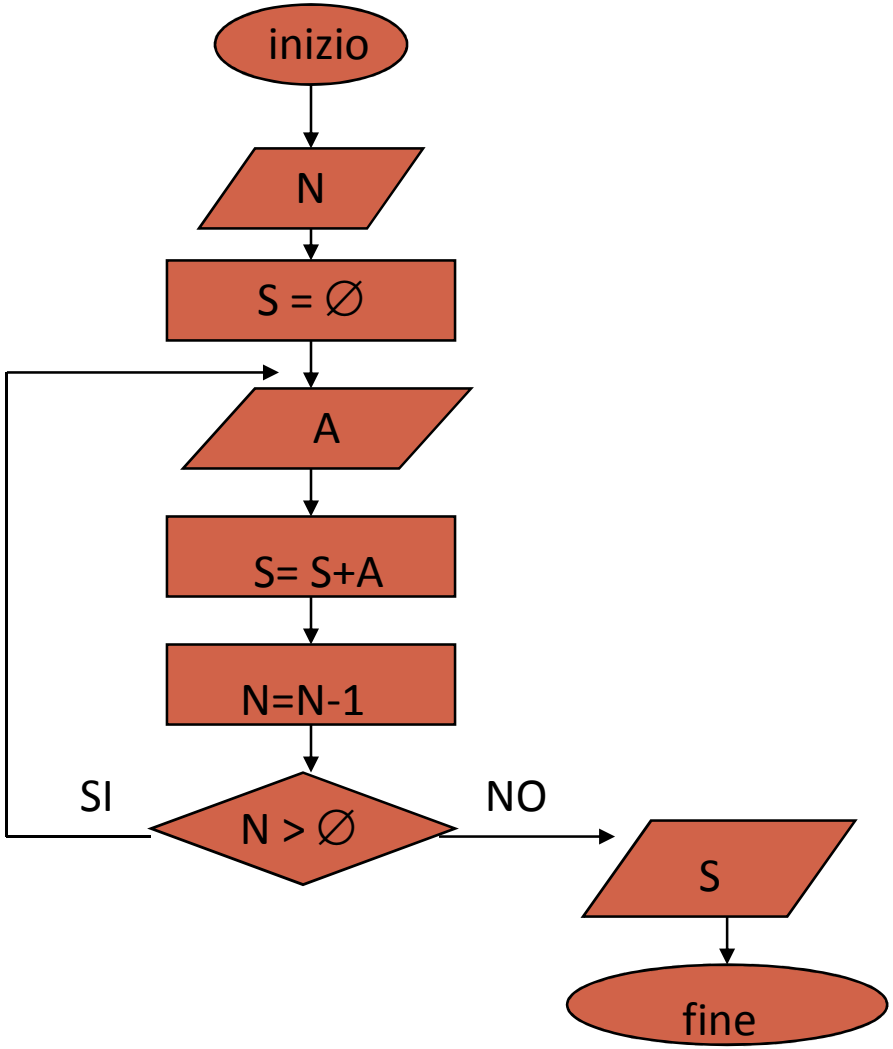
Diagramma a blocchi o flow-chart

Ha il pregio di evidenziare visivamente l'avanzamento in sequenza e le varie strutture che compongono l'algoritmo, presenta istruzioni di input e/o output, calcolo e/o di elaborazione, condizioni ed individua un inizio ed una **fine**.



ESEMPIO DI ALGORITMO RAPPRESENTATO CON IL FLOW-CHART

Somma S di una sequenza di N numeri di valore A *variabile*



Linguaggio di progetto o pseudo-codifica

È un linguaggio formale (**linguaggio di progetto**), con regole prive di ambiguità ed eccezioni che esprimono i vari tipi di istruzioni. Viene definito

pseudo-codifica

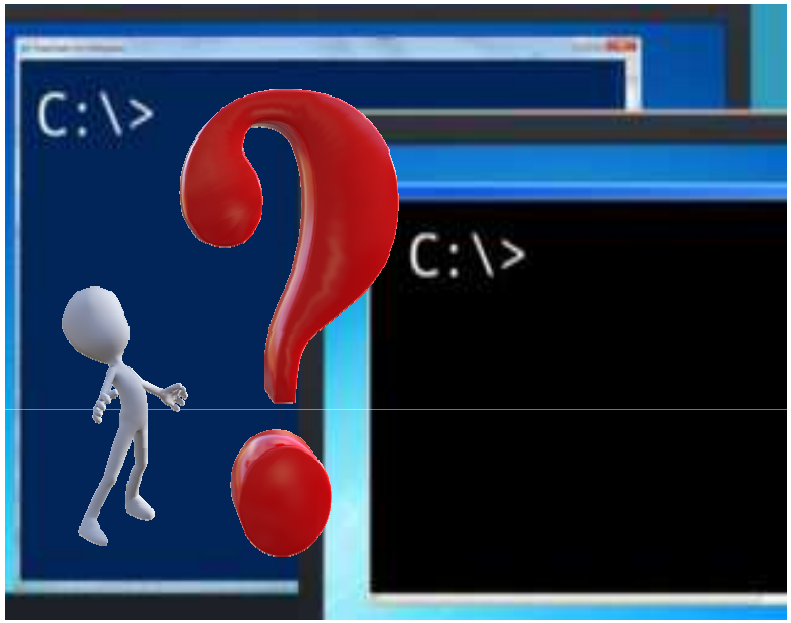
```
inizio
    leggi N
    S ← ∅
    fai
        leggi A
        S ← S+A
        N ← N-1
    mentre ( N > ∅ )
    stampa S
fine
```

al posto di **inizio/fine** si possono usare le **parentesi graffe**

MODULO 2 - PROMPT DEI COMANDI

PROMPT DEI COMANDI

Il **PROMPT DEI COMANDI**, è una SHELL (interprete dei comandi) di Windows che permette di digitare ed eseguire i comandi di DOS dalla linea di comando.



Utilizzare il promt dei comandi rende *più veloce* l'esecuzione dei comandi stessi perché baipassano l'interfaccia grafica

PROMPT: è la sequenza di caratteri che indica che il computer è in attesa di comandi, ed è rappresentata con questa sequenza:
nome dell'unità a disco
:
\
>
Quindi **C:\>**
Indica che ci troviamo nella *root* (radice).

FILE E DIRECTORY

I files sono identificati da un nome di max 8 caratteri seguito da .estensione ovvero ¾ caratteri che ci ricordano il tipo del file o il programma che lo ha generato.

Esempio prova.txt → file di testo

esercizio.cpp → file contenente codice C++

cmd.exe → file eseguibile che lancia il prompt dei comandi

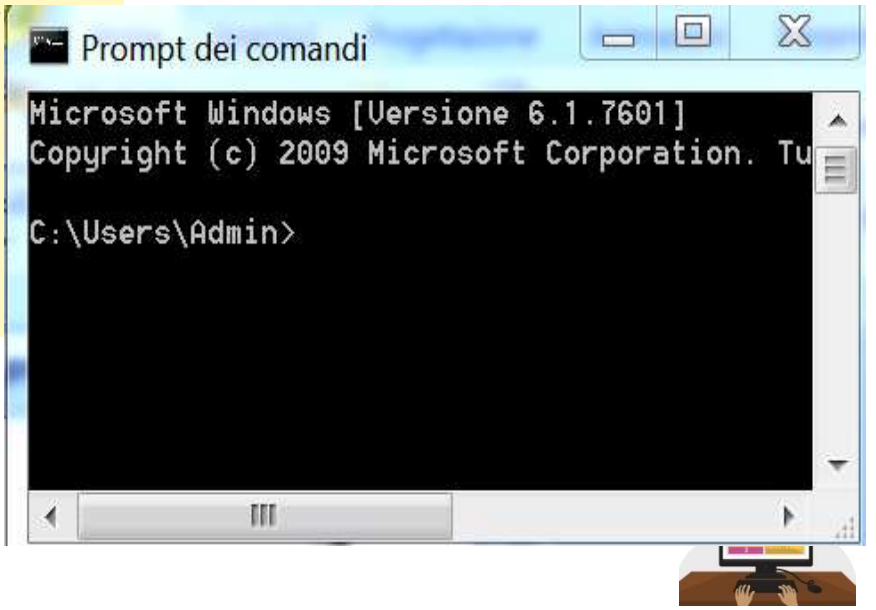
miosito.html → pagina web

Per fare riferimento ai files, dobbiamo indicare oltre al nome e all'estensione, il percorso (***PATHNAME***) che indica dove questo si trova.

Esempio**: C:\users\3D\esercizi>**

Esempio: l'immagine ci dice che stiamo visualizzando i files contenuti nella directory Admin che si trova nella directory Users nell'unità C (hard disk)

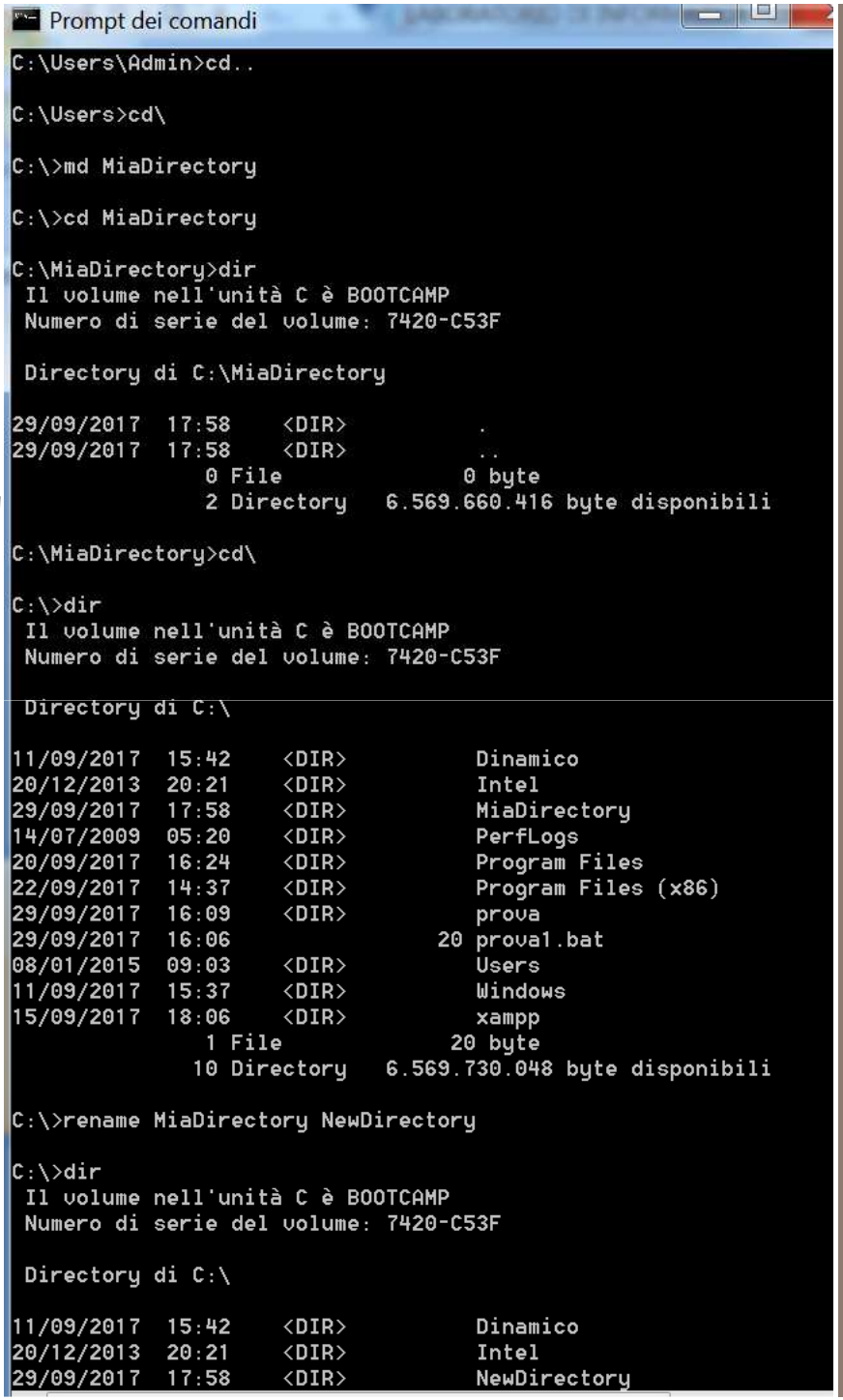
I files sul disco sono organizzati secondo una **struttura logica** ad albero, contenente varie directory e file, ogni directory a sua volta può contenere altrettante directory e files.
La directory principale è definita ***radice*** o ***root***



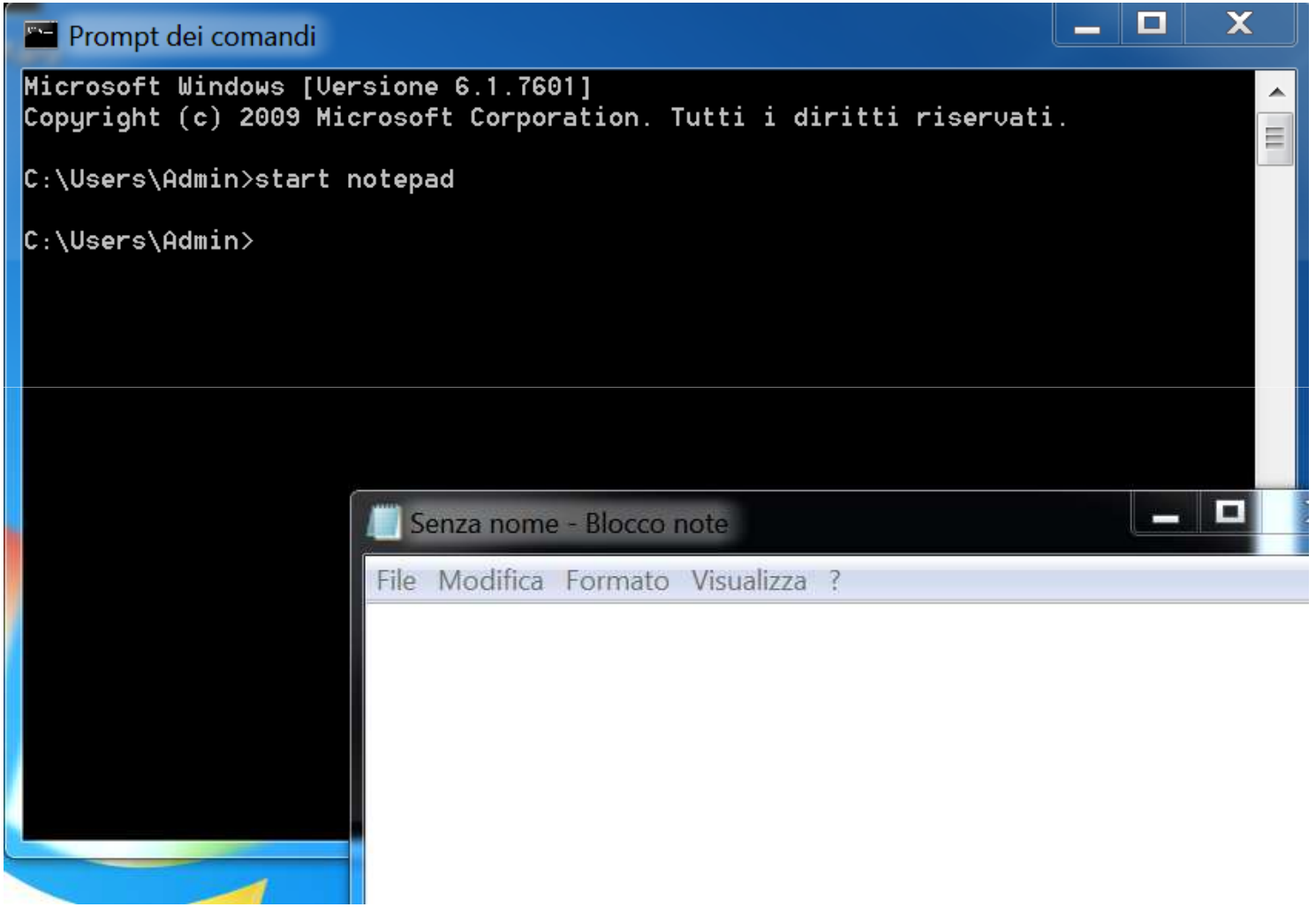
ESERCITIAMOCI

Per avviare il prompt dei comandi, possiamo procedere in diversi modi:

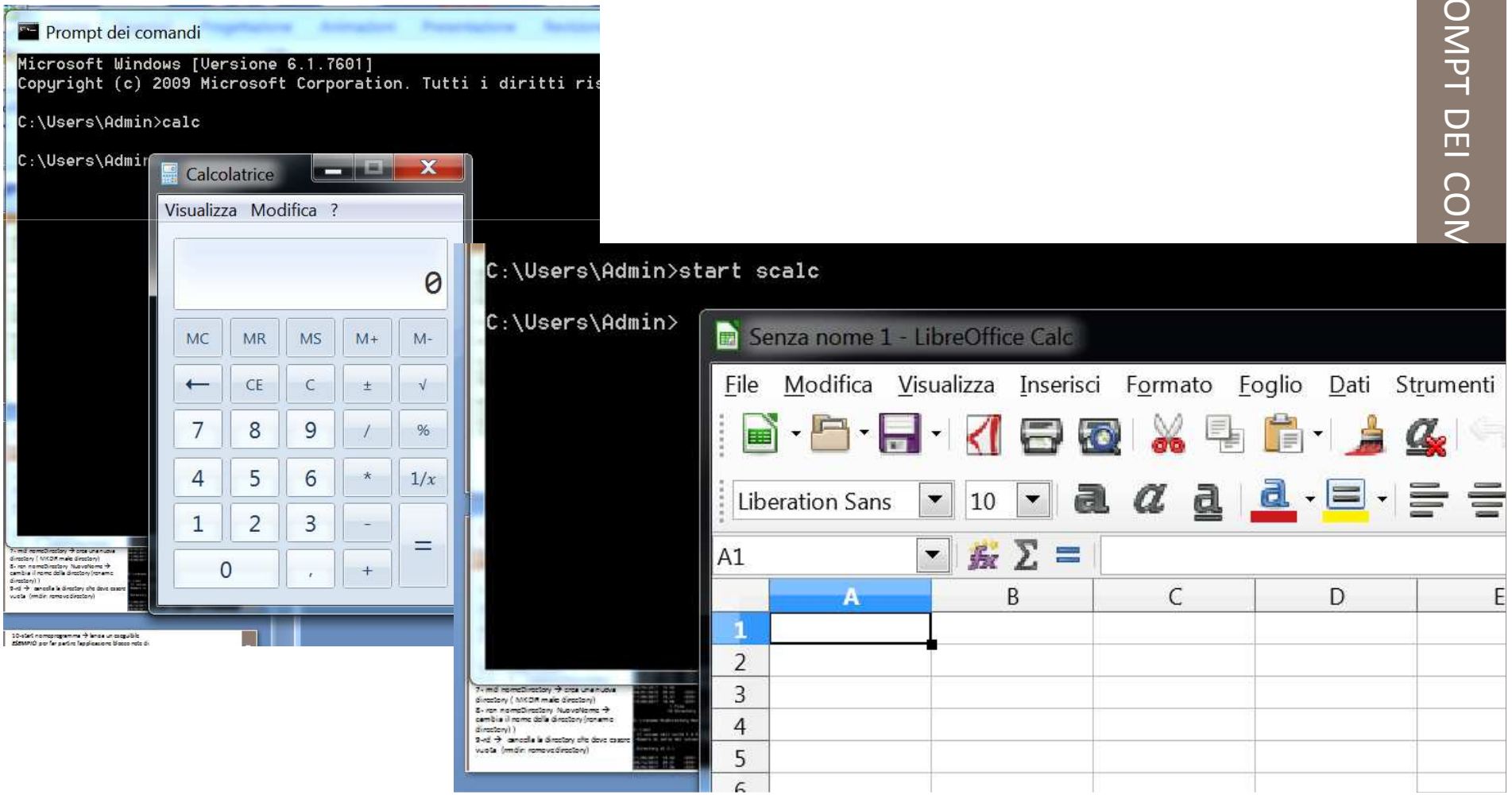
1- utilizzare la funzione Ricerca di Windows scrivendo "prompt dei comandi"
2 -utilizzare la funzione ESEGUI di Windows digitando il nome del file eseguibile "cmd.exe"
3- avviare il prompt dei comandi dal menu start – tutti i programmi- accessori – prompt dei comandi.
4 - cd.. → Torna alla directory precedente
5 - cd\ → torna alla directory radice (CHDIR change directory)
6- dir → visualizza elenco directory
7- md nomeDirectory → crea una nuova directory (MKDIR make directory)
8- ren nomeDirectory NuovoNome → cambia il nome della directory (rename directory))
9-rd → cancella la directory che deve essere vuota (rmdir: remove directory)



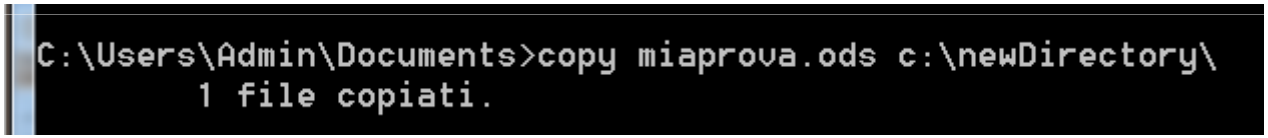
10-start nomeprogramma → lancia un eseguibile
ESEMPIO per far partire l'applicazione blocco note di Windows digitiamo ***start notepad***



Ovviamente per avviare un programma dalla linea di comando è necessario conoscere il nome esatto del file eseguibile .exe o bat.
Un altro esempio infatti potrebbe essere questo:
Il programma CALC della suite Libre Office, non si avvia digitando ***CALC*** come si potrebbe pensare, piuttosto questo comando avvia l'applicazione CALCOLATRICE di Windows. Per avviare Calc di Libre Office, il file eseguibile è ***scalc.exe***

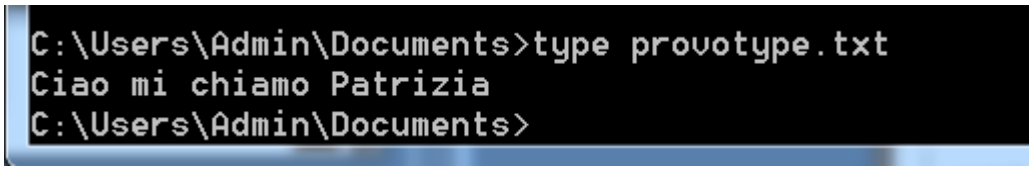


11) Ora che sappiamo avviare un'applicazione, creiamo un foglio di calcolo, file .ods (generato con CALC), oppure un documento di testo, file .txt (generato con notepad) e salviamolo nella cartella documenti.
Con gli appositi comandi DIR e CD spostiamoci da una directory all'altra e verifichiamo la presenza del nostro documento.
12) ora per testare un altro comando del DOS, copiamo questo file dalla directory documenti alla directory NewDirectory che abbiamo creato precedentemente e sempre con i comandi cd e dir verifichiamo. Per copiare usiamo
copy nomefile c:\Newdirectory
NB: se il file da copiare non si trova nella directory corrente è necessario indicarne il pathname

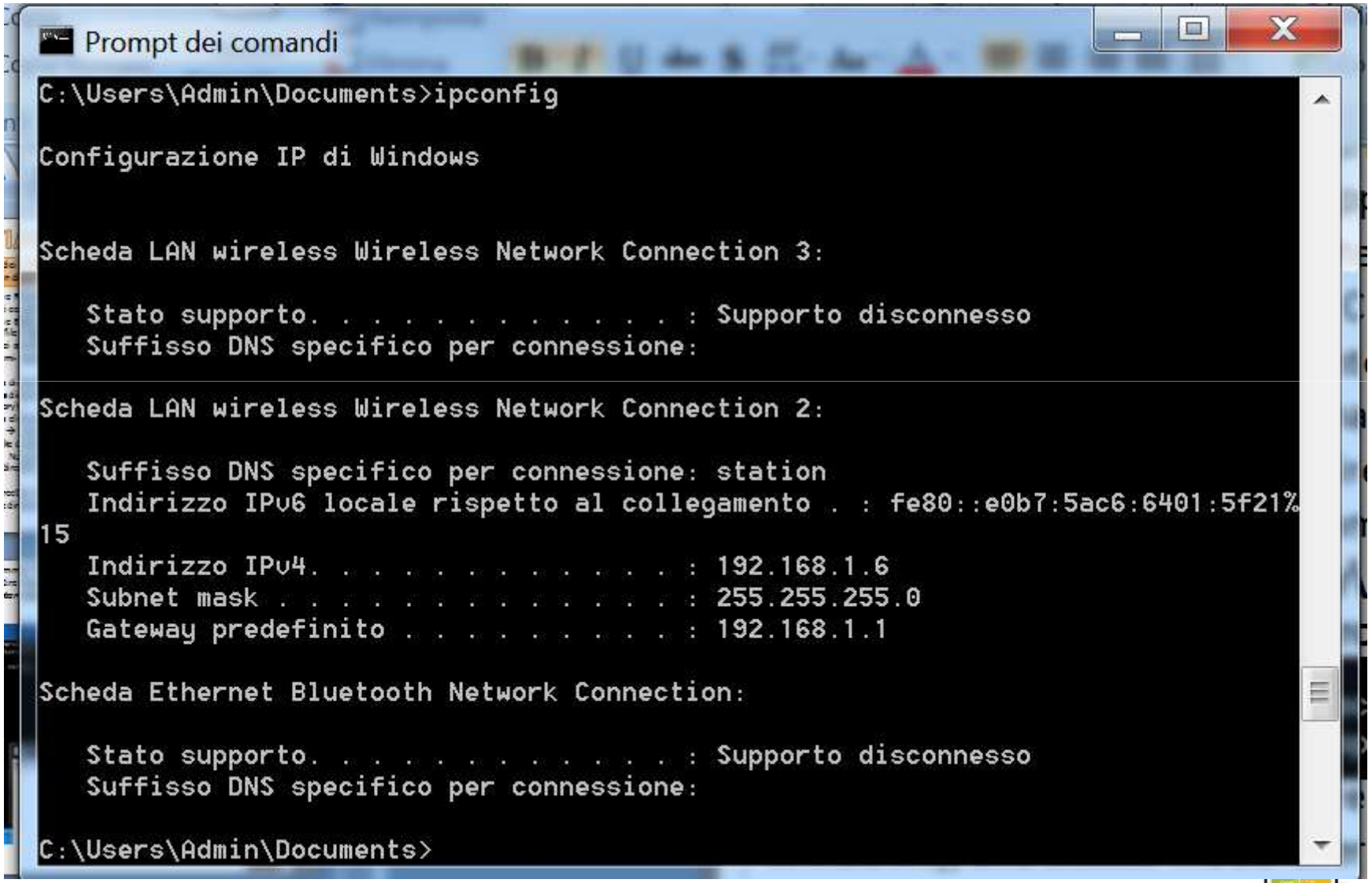


13) se invece dobbiamo spostare un file da una directory ad un'altra (taglia e incolla di Windows), allo stesso modo useremo **move nomefile c:\Newdirectory**

14) Con il comando ***type nomeFile*** visualizziamo il contenuto di un file di testo.



15) usiamo il comando ***ipconfig*** per conoscere il numero ip del nostro pc



16) Mentre Remove cancella una directory solo se questa è vuota, il comando ***del nomefile*** cancella il file. Dopo aver creato un file di prova, cancellarlo.

17) Nel riferirsi ai files, possiamo utilizzare i CARATTERI JOLLY ovvero:

* → indica qualsiasi carattere per qualsiasi numero di caratteri
es: *.txt (pippo.txt, lettura.txt,a.txt , ecc.)
? → indica qualsiasi caratter per un carattere
es: ?.txt (a.txt,g.txt)

L'uso dei caratteri jolly è utile quando dobbiamo ricercare file sul pc abinandone l'uso al comando dir al quale possiamo abinare anche
il parametro /p
per visualizzare una pagina alla volta l'elenco delle directory.

C:\Users\Admin\Desktop\somma.bat - Notepad++

File Modifica Cerca Visualizza Formato Linguaggio Configurazione Strumenti M

prova1.bat | stampa.java | somma.class | somma.java | stampa.java

```
1  @ECHO OFF
2  set /A x=%1
3  set /A y=%2
4  set /A z=x+y
5  echo ----------------------------------------
6  echo ADDIZIONE: %x% + %y% = %z%
7  echo ----------------------------------------
```

Prompt dei comandi

```
C:\Users\Admin\Desktop>somma.bat 9 221
----------------------------------------
ADDIZIONE: 9 + 221 = 230
----------------------------------------

C:\Users\Admin\Desktop>
```

MODULO 3 - LINGUAGGI DI PROGRAMMAZIONE C - C++

STRUTTURA DI UN PROGRAMMA

Commenti su più righe

C++

Le righe che iniziano con il simbolo **#** non contengono istruzioni ma indicazioni per il compilatore. **iostream,** è l'**header file** (**file di intestazione**) che permette al compilatore di importare le funzioni della libreria standard di **I/O** necessaria in C++ per gestire l'interazione con l'utente, che contiene, ad esempio, **cin** (per l'acquisizione dell'input da tastiera) e **cout** (per l'emissione dell'output a video)

```
/*----------------------------------------------
  Nome del programma:
  Autore:
  Data:
  Descrizione:

----------------------------------------------*/
#include <iostream>
using namespace std;


//------------------------------ main --------
int main()
{

   return 0;
}
```

I namespace (o spazio dei nomi) sono contenitori di nomi. **namespce std** contiene gli identificatori standard del C++

C

```
/*----------------------------------------------
  Nome del programma:
  Autore:
  Data:
  Descrizione:

----------------------------------------------*/
#include <stdio.h>


//------------------------------ main --------
int main()
{

   return 0;
}
```

stdio.h, sta per "standard input-output header", è **l'header file** della libreria standard del C che contiene le funzioni e i tipi usati per le varie operazioni di **I/O**

Commento su una riga: questo è possibile inserirlo anche a seguito di un istruzione per rappresentare il significato dell'istruzione stessa

la separazione tra istruzioni è indicata dal punto e virgola ; che termina ciascuna di esse. La suddivisione del programma in più righe serve a rendere il programma più leggibile alle persone che lo leggono, per il compilatore la cosa non fa' alcuna differenza. Fanno eccezione le righe di commento e le righe della dichiarazione di inclusione.

C++

```
/*----------------------------------------------------
  Nome del programma:
  Autore:
  Data:
  Descrizione:

----------------------------------------------------*/
#include <iostream>
using namespace std;



//------------------------------ main ----------------
int main()
{

    return 0;
}
```

C

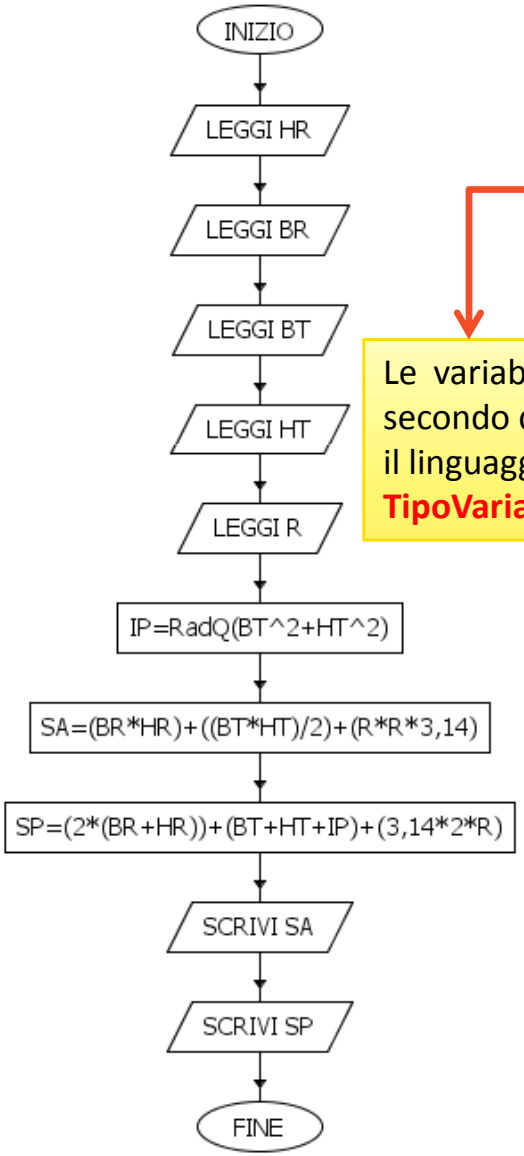
```
/*----------------------------------------------------
  Nome del programma:
  Autore:
  Data:
  Descrizione:

----------------------------------------------------*/
#include <stdio.h>



//------------------------------ main ----------------
int main()
{

    return 0;
}
```

Indica l'inizio del programma vero e proprio che viene visto come una funzione pertanto rispetta la sintassi:
Tipo_dato_restituito
nome_funzione ()

Per entrambi i linguaggi: tra le parentesi graffe { } la sequenza di istruzioni

Una funzione restituisce un valore come risultato dell'elaborazione: 0 (Falso) , diverso da 0 (Vero). Return 0 sta per errore = Falso, ovvero non si sono verificati errori.

Calcolare la somma delle aree e la somma dei perimetri di un rettangolo , un triangolo rettangolo e un cerchio



DATI VARIABILI			
Nome	**Tipo**	**Uso**	**Descrizione**
BR	Num. reale	Input	Base del rettangolo
HR	Num. reale	Input	Altezza del rettangolo
BT	Num. reale	Input	Base del triangolo rettangolo
HT	Num. reale	Input	Altezza del triangolo rettangolo
R	Num. reale	Input	Raggio del cerchio
IT	Num. reale	Lavoro	Ipotenusa del triangolo, necessaria per calcolo perimetro
SA	Num. reale	Output	Somma delle aree
SP	Num. reale	Output	Somma dei perimetri

DATI COSTANTI		
Valore	**Tipo**	**Descrizione**
3.14	Num. reale	Valore di Pi-Greco necessario per calcolare area e perimetro del cerchio

Le variabili vanno dichiarate secondo questa sintassi sia per il linguaggio **C** che per il **C++:**
TipoVariabile nomeVariabile

Per acquisire i dati di input:

C++	**C**
cin>>variabile	**scanf("%lf", &variabile)**

Per rilasciare i dati di output:

C++	**C**
cout >> variabile	**printf("%lf\n", variabile)**

INIZIO
Leggi BR
Leggi HR
Leggi BT
Leggi HT
Leggi R
Calcola IP = Radice quadrata di (BT^2 + HT^2)
Calcola SA = (BR*HR) + ((BT*HT)/2) + (R*R*3.14)
Calcola SP = (2*(BR+HR)) + (BT+HT+IP) + (3.14*2*R)
Scrivi SA
Scrivi SP
FINE

C++

```
/*--------------------------------------------------------------------------------
  Nome del programma: Somma Aree e Perimetri
  Autore:   Patrizia
  Data:     19 ottobre 2017
  Descrizione: Somma aree e perimetri di un rettangolo, un triangolo rettangolo e un cerchio

--------------------------------------------------------------------------------*/
#include <iostream>
#include <math.h>
using namespace std;

int main(int argc, char *argv[])
{
  double BR, BT, HR, HT, R,IP, SA, SP;
  cout<<"Inserisci altezza rettangolo: "<<endl;
  cin >> HR;
  cout<<"Inserisci base rettangolo: "<<endl;
  cin >> BR;
  cout<<"Inserisci base triangolo: "<<endl;
  cin >> BT;
  cout<<"Inserisci altezza triangolo: "<<endl;
  cin >> HT;
  cout<<"Inserisci il raggio del cerchio: "<<endl;
  cin >> R;
  IP= sqrt((BT*BT)+(HT*HT));
  SA=(BR*HR)+((BT*HT)/2)+(R*R*3,14);
  SP=(2*(BR+HR))+(BT+HT+IP)+(3,14*2*R);
  cout <<" la somma delle aree è: "<<SA<<endl;
  cout<<" la somma dei perimetri è: "<<SP<<endl;
  cout << SP << endl;

  return 0;
```

C

```
/*--------------------------------------------------------------------------------
  Nome del programma: sommaAreaPerim.c
  Autore: Patrizia
  Data: 19 ottobre 2017
  Descrizione: Somma delle aree e dei perimetri di un rettangolo
               un triangolo rettangolo e un cerchio.
--------------------------------------------------------------------------------*/
#include <stdio.h>
#include <math.h>
int main(int arc, char *argv[])
{
  double BR, BT, HR, HT, IP, R, SA, SP;

  puts( "inserisci l'altezza del rettangolo: ");
  scanf("%lf", &HR);
  puts ("inserisci la base del rettangolo: ");
  scanf("%lf", &BR);
  puts ("inserisci la base del triangolo: ");
  scanf("%lf", &BT);
  puts ("Inserisci l'altezza del triangolo: ");
  scanf("%lf", &HT);
  puts ("Inserisci il raggio: ");
  scanf("%lf", &R);
  IP=sqrt(BT*BT)+(HT*HT);
  SA=(BR*HR)+((BT*HT)/2)+(R*R*3,14);
  SP=(2*(BR+HR))+(BT+HT+IP)+(3,14*2*R);
  puts("La somma delle aree è: ");
  printf("%lf\n", SA);
  puts("la somma dei perimetri è: ");
  printf("%lf\n", SP);

  return 0;
```

C++	C
DICHIARAZIONE VARIABILI	
char x; int y; float z; double t; char nome[30];	char x; int y; float z; double t; char nome[30];
LEGGI	

cin >> x;
cin >> y;
cin >> z;
cin >> t;
cin >> nome;
	scanf("%c", &x);
	scanf("%d", &y);
	scanf("%f", &z);
	scanf("%lf", &t);
	scanf("%s", nome);

SCRIVI	

cout << x;
cout << y;
cout << z;
cout << t;
cout << nome;
	printf("%c", x);
	printf("%d", y);
	printf("%f", z);
	printf("%lf", t);
	printf("%s", nome);

[C++] cout<<"x="<<x<<" y="<<y<<" t="<<t<<"nome="nome<<endl;	
[C] printf("x=%c y=%d z=%f t=%lf nome=%s\n",x, y, z, t, nome);	
[C] printf("x=%c x=%d x=%X", x, x, x);