Corso "Basi di Informatica A e B"
Diario delle Lezioni, delle Esercitazioni e del Tutorato

Università di Torino
Corso di Laurea in Matematica
A.A. 2016-2017

Docente: Stefano Berardi

Tutore: Giuseppe Corrente
Assistente: Chiara Balestra

Presentazione del Corso di Basi di Informatica del 2017 - Corso di Laurea in Matematica dell'Università di Torino

Docente: Stefano Berardi

Un corso di Basi di Informatica fa parte del Corso di Laurea in Matematica dell'Università di Torino. È necessario spiegare perché? Per la maggior parte dei corsi non c'è alcuna necessità. Voi non vi preoccupate di sapere perché questo Corso di Laurea contiene un corso di Analisi, oppure uno di Geometria o uno di Algebra: vi aspettate che siano stati scelti in base alla loro importanza per il Corso di Laurea che state seguendo. Il corso di Basi di Informatica, tuttavia, tratta di un argomento non strettamente collegato con la Matematica. Vale quindi la pena di spendere qualche parola per spiegare come mai trovate un corso di Basi tra i corsi del primo anno.

0. Perché un corso di Basi di Informatica.

Il nome scelto per il corso, "Basi di Informatica", non è così preciso: sarebbe più corretto chiamare questo corso **"Basi di Programmazione"**. L'Informatica infatti è una scienza che tratta, per esempio, di Metodi Formali, Intelligenza Artificiale, Data Mining, riconoscimento di immagini e parole, Traduzione Automatica, Teoria della Complessità e molto, molto altro. Di tutto questo nel corso di Basi non faremo parola: il corso tratta invece di Basi di Programmazione. La Programmazione è uno strumento che viene utilizzato in diverse scienze, Matematica compresa, non solo in Informatica.

Ci sono vari motivi per includere un corso di Basi di Programmazione in un corso di Laurea in Matematica. Innanzitutto, l'esperienza dimostra che molti matematici trovano lavoro come programmatori: dunque saper programmare è un titolo per cercare lavoro, una carta in più che uno può giocare. Da un punto di vista più strettamente scientifico, invece, la programmazione è utilizzata in molte scienze per creare modelli virtuali di quello che uno sta studiando. Per esempio, in Fisica si creano modelli virtuali del sistema solare o del plasma, e così via. In Biologia si creano modelli virtuali della propagazione di un virus o di un

batterio o della crescita di una colonia di cellule. Lo stesso vale per l'Economia e per molte altre scienze di tipo matematico.

A tutte queste osservazioni, importanti ma ben note, volevo aggiungere qui che la programmazione è uno strumento importante per la Matematica stessa. Innanzitutto per la matematica applicata: per l'Analisi, il Calcolo Numerico, la Statistica e così via. Tuttavia, e questo fatto è meno conosciuto, la programmazione è importante anche per il lavoro del matematico teorico: vediamo qualche esempio.

1. La probabilità che due interi positivi siano primi tra loro.

Immaginiamo di porci il problema di conoscere qual'è la probabilità che due numeri interi positivi scelti a caso siano primi tra loro.

Come può aiutarci la programmazione? Prima di cercare di risolvere questo problema possiamo scrivere un programma e passare in esame un gran numero di coppie di interi: per esempio, le prime 100 coppie, le prime 10 mila, il primo milione o i primi 100 milioni di coppie, per vedere in che percentuale queste coppie di numeri sono primi tra loro. Quello che otteniamo è che circa il 61% delle coppie esaminate sono prime tra loro, e questa percentuale tende a stabilizzarsi verso valori via via più precisi al crescere del campione in esame.

Che cosa abbiamo ottenuto? Questo esame empirico evidentemente non può consentirci di concludere che esista una probabilità che due numeri siano primi tra loro: potrebbe essere che cambiando il campione esaminato la probabilità salga e scenda senza mai stabilizzarsi verso un valore preciso. Tuttavia, il fatto che dopo molte prove ma il risultato ottenuto tenda a stabilizzarsi verso il valore di 0,61 ci fa congetturare che questa sia effettivamente la probabilità che due numeri siano primi tra loro, almeno all'incirca. Non è abbastanza per essere sicuri, ma è abbastanza perché valga la pena di fare uno studio teorico. Adesso, grazie alla verifica empirica fatta, consideriamo verosimile che l'evento "due numeri sono primi tra loro" abbia una probabilità ben definita e che questa probabilità sia all'incirca il valore che abbiamo trovato.

Inoltre scrivere un programma aiuta a porci nuove domande. In questo caso, potrebbe venirci la curiosità di sapere qual è la probabilità che due numeri abbiano massimo comun divisore non più

1, come per i numeri primi tra loro, ma 2 oppure 3 oppure qualunque valore intero. Avendo un programma a disposizione, è piuttosto semplice cambiare lo studio statistico e controllare non più quali coppie di numeri interi abbiano massimo comun divisore 1 ma quanti abbiano massimo comun divisore 2 oppure 3 oppure un qualunque altro valore. Dopo un po' di prove, scopriamo che se la probabilità di avere un massimo comun divisore 1 vale circa 0,61, la probabilità di avere un massimo comun divisore 2 vale circa un 1/4 di questo valore, la probabilità di avere un massimo comun divisore 3 vale circa 1/9 di questo valore, e così via.

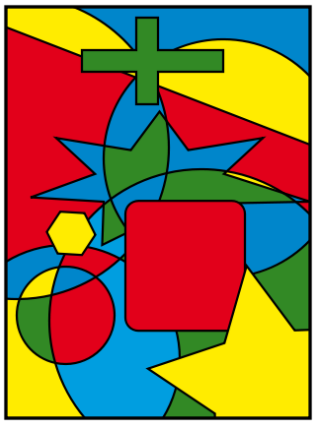
In altre parole, dopo un po' di verifiche empiriche ci viene naturale congetturare qualcosa a cui non avevamo pensato: in questo caso, che la probabilità di un massimo comun divisore sia inversamente proporzionale al quadrato del massimo comun divisore stesso. A questo punto se chiamiamo P il la probabilità che due numeri abbiano massimo comun divisore 1, la probabilità che due numeri abbiano massimo comun divisore 2 sarà P/4, la probabilità che abbiano massimo comun divisore 3 sarà P/9 e così via. La probabilità che un intero abbia massimo comun divisore 1 oppure 2 oppure 3 eccetera vale 1, quindi 1 sarà uguale ad (P + P/4 + P/9 + …). In altre parole ci viene da congetturare che 1 = P*(1 + 1/4 + 1/9 + …) e dunque che P = 1/(1 + 1/4 + 1/9 + …). Consultando un testo di Analisi, possiamo verificare che (1 + 1/4 + 1/9 + …) = $\pi^2/6$ (questo risultato è noto come *Problema di Basilea*). Quindi partendo solo dalle verifiche empiriche siamo arrivati a formulare una congettura precisa sul valore di P: **«P vale $6/\pi^2$»**, dove $6/\pi^2$ = 0,607927 … . Adesso sta noi usare le nostre conoscenze matematiche per stabilire se è vero no. Effettivamente si sa provare che P = $6/\pi^2$ (è un problema ben noto di Teoria dei Numeri).

Abbiamo visto una interazione tra uno studio teorico e una verifica empirica, basata sulla costruzione di mondi virtuali e fatta attraverso la programmazione. In genere una verifica empirica fa nascere una congettura teorica, viceversa una congettura teorica motiva una nuova verifica empirica e così via, e c'è una continua interazione tra i due aspetti, che tuttavia restano ben distinti. Una verifica empirica è molto più facile da realizzare e costa poco tempo, ma può trattare solo un numero finito di casi e quindi non può darci una certezza su enunciati che riguardino un numero infinito di oggetti. Invece una prova matematica fornisce una certezza anche per enunciati su un numero infinito di oggetti, qualcosa che empiricamente non potremo mai controllare alla perfezione. Però una prova matematica è molto più difficile da ottenere. Quindi è bene che la ricerca di una prova

sia diretta da verso gli enunciati che in seguito a verifiche empiriche ci appaiono più plausibili di altri.

2. Il Teorema dei Quattro Colori.

Ci sono enunciati matematici che riguardano l'infinito, ma che curiosamente si possono ridurre a enunciati che riguardano un numero finito di casi, e che quindi possono venire controllati attraverso un programma. L'esempio più famoso esempio riguarda il Teorema dei Quattro Colori, che dice: data una qualunque cartina piana è possibile colorare le regioni di questa cartina usando 4 colori, in modo tale che non ci siano mai due regioni con lo stesso colore che abbiano in comune un tratto di linea. Come nell'esempio qui sotto, preso da Wikipedia:



Non esistono dimostrazioni puramente matematiche di questo risultato, ma esiste una prova che combina matematica e programmazione. Nel 1977, Kenneth Appel e Wolfgang Haken, due matematici dell'Università dell'Illinois, sono stati capaci di dimostrare che esistono circa 2000 cartine finite (poi ridotte a circa 1500), tali che se esiste un controesempio alla congettura dei quattro colori, allora esiste un controesempio in questo insieme di 2000 cartine. Di conseguenza, per verificare la congettura è sufficiente scrivere un programma che esamina queste 2000 cartine e controlla se per ognuna di esse c'è una colorazione che utilizzi soltanto 4 colori oppure no. Questo programma è stato scritto e la verifica ha mostrato che queste 2000 cartine possono venire colorate con soltanto 4 colori non adiacenti: di conseguenza la congettura dei quattro colori vale. A tutt'oggi non esistono prove della congettura dei quattro colori puramente matematiche: tutte le prove combinano una parte di ragionamento matematico con un programma che verifica un numero finito di casi. Notiamo che in questo caso non c'è stato interazione tra l'aspetto

di programmazione e l'aspetto di teoria: la teoria ha suggerito un controllo da fare, ma questo controllo purtroppo non ha suggerito nuovi enunciati teorici da verificare per migliorare la prova.

3. La congettura di Keplero.

Ci sono altri esempi simili, per esempio la congettura di Keplero, conosciuta anche come diciottesimo problema di Hilbert. La lista dei 23 problemi di Hilbert è una lista dei problemi considerati più difficili della matematica. La congettura di Keplero dice che la disposizione di sfere di ugual diametro che realizza la massima densità possibile è quella esagonale, ottenuta facendo crescere all'infinito lungo in tutte le direzioni la disposizione seguente:

Nel 1998 Thomas Hales, oggi professore all'università di Pittsburgh, dimostrò che la congettura di Keplero poteva essere ridotto al problema di controllare se 100 mila sistemi di disequazioni lineari avevano una soluzione di un certo tipo. Quello che il matematico fece, dopo aver ridotto la congettura di Keplero a numero finito di verifiche, fu scrivere un programma che controllasse che le 100 mila disequazioni lineari avessero una soluzione del tipo richiesto. La verifica diede risultato positivo: questo diede il secondo esempio importante di una prova che non appartiene unicamente né alla matematica né alla programmazione ma usa una combinazione dei due strumenti. Anche in questo caso è purtroppo mancato l'interazione tra la programmazione e lo studio teorico. Lo studio delle 100 mila disequazioni non ha fornito idee su come migliorare la prova della congettura di Keplero.

4. La congettura di Collatz.

Aggiungiamo un ultimo esempio per ricordare che la verifica empirica e lo studio teorico, per quanto strettamente collegati, restano ben distinti. Parleremo della congettura di Collatz. Prendiamo un qualunque numero intero positivo, se è pari

dividiamolo per 2, se è dispari moltiplichiamo per tre e aggiungiamo 1, e ripetiamo queste operazioni fino a raggiungere il valore 1. Per esempio, partendo da 7 otteniamo:

7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 1

La congettura di Collatz dice: se ripetiamo queste due operazioni, dopo un numero finito di passi raggiungiamo sempre 1. **A tutt'oggi la congettura di Collatz è aperta.** Paul Erdős, il miglior studioso di Matematica Combinatoria del ventesimo secolo, concluse che «**la Matematica non è ancora pronta per questo tipo di problemi**».

La congettura di Collatz è stata verificata con un programma per i primi 5 miliardi di miliardi di numeri interi (trovate i dettagli su Wikipedia). Per tutti questi valori si è trovato che alla fine la successione definita sopra raggiunge il valore 1. Questo tuttavia non è sufficiente a dire che la congettura di Collatz è vera: innanzitutto perché 5 miliardi di miliardi di interi non sono tutti i numeri interi; e poi perché **si è già notato che congetture che apparivano plausibili ed erano molto semplici si sono rivelate false per dei valori davvero molto grandi**. Di conseguenza, il fatto che una congettura di Collatz sia vera per 5 miliardi di miliardi di numeri interi non basta nemmeno a renderla probabile, ma almeno ci dice che è inutile cercare un controesempio tra i numeri piccoli.

In ognuno dei casi considerati, comunque, la programmazione si è rilevato uno strumento importante e complementare alla ricerca matematica, qualcosa che deve far parte, almeno a livello di basi, della conoscenza di ogni matematico.

Organizzazione del Corso "Basi di Informatica"

1. **Il programma del corso** è: istruzioni e definizioni dei linguaggi di programmazione: ***input/output, condizionale, funzioni, ricorsione, cicli while;*** e rappresentazione dei dati: ***stringhe, strutture, vettori e matrici***. Questo programma corrisponde ai primi 10 capitoli (circa 100 pagine) del libro di testo ***«How to think like a computer scientist»***, disponibile gratuitamente on-line all'indirizzo:
   http://greenteapress.com/thinkcpp/index.html
   Leggere le dispense che avete in mano dovrebbe essere sufficiente per il corso: tuttavia, se avete l'impressione di non aver capito una lezione rileggetela sul libro. La presentazione delle matrici non viene ripresa dal libro di testo, ma si trova nella sezione dedicata alla Settimana 11. Sempre nella Settimana 11, trovate la libreria ***"Matrix.cpp"***, che non fa parte del C++ e che quindi dovrete ricopiare.

2. Il sito Moodle del corso di Basi nel 2016/2017 è:

   http://math.i-learn.unito.it/course/view.php?id=808

A queste dispense hanno contribuito nel 2013/2014: *Stefano Berardi, Cristina Battaglino, Silvia Steila e Ugo de' Liguoro*, docente del corso "Basi di Informatica" B, a cui si aggiungono, nel 2014/2015: *Alessio Antonini e Luca Canensi*.

- In genere il materiale di ogni lezione è diviso in tre parti, che contengono:
  a. Gli **esempi** tratti da capitoli del libro di testo,
  b. Gli **esercizi** proposti in classe,
  c. Le **soluzioni** per questi esercizi.
- A volte la lezione è fatta di soli esercizi, a volte inizia con un ripasso di lezioni precedenti.
- Ogni tutorato consiste di esercizi, che di solito riguardano le due lezioni della settimana appena trascorsa. Il **Tutorato 1** richiede di aver svolto le due lezioni della **Settimana 1** e così via.
- Per ogni esercizio del corso oppure del tutorato vengono inclusi testo e soluzioni: le dispense vere e proprie occupano uno spazio ridotto. Per restare in pari con il corso consigliamo di svolgere in classe ***almeno metà degli esercizi proposti***, e di finire il resto a casa.

Indice del Corso

Il corso è diviso in 12 settimane. Ogni settimana di corso corrisponde a una sezione del Diario delle Lezioni, consiste di due lezioni e termina con un Tutorato: i dettagli dipendono dal calendario di ogni anno. L'unico programma di una certa lunghezza a cui accenniamo nel corso, una animazione di spirali, trova posto in appendice: non viene richiesto all'esame.

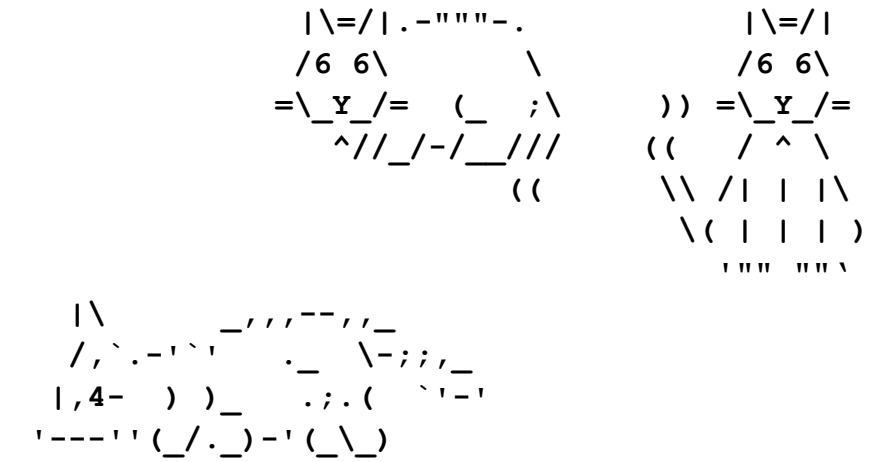
Ricordiamo un'ultima volta che la presentazione delle **matrici** non viene ripresa dal libro di testo, ma si trova nella **Settimana 11.** All'inizio della sezione dedicata alla Settimana 11, trovate la libreria ***"Matrix.cpp"***, che utilizzeremo negli esercizi sulle matrici: non fa parte del C++, quindi dovrete ricopiarla.

Settimana 01. Lezioni 01 e 02: **output** e variabili.
Tutorato 01: **output** e variabili.
Settimana 02. Lezioni 03 e 04: funzioni e **condizionale.**
Tutorato 02: funzioni e **condizionale.**
Settimana 03. Lezioni 05 e 06: funzioni con **valore di ritorno.**
Tutorato 03: funzioni con **valore di ritorno.**
Settimana 04. Lezioni 07 e 08: cenni di **ricorsione.**
Tutorato 04: cenni di **ricorsione.**
Settimana 05. Lezioni 09 e 10: ricorsione cicli **while.**
Tutorato 05: ricorsione cicli **while.**
Settimana 06. Lezioni 11 e 12: esercizi sui cicli **while.**
Tutorato 06: esercizi sui cicli **while.**
Settimana 07. Lezioni 13 e 14: **stringhe.**
Tutorato 07: ricorsione e **stringhe.**
Settimana 08. Lezioni 15 e 16: **strutture.**
Tutorato 08: **strutture.**
Settimana 09. Lezioni 17 e 18: **vettori.**
Tutorato 09: **strutture** (ripasso).
Settimana 10. Lezioni 19 e 20: esercizi sui **vettori.**
Tutorato 10: **vettori.**
Settimana 11. Lezioni 21 e 22: **matrici.** Libreria ***"Matrix.cpp"***.
Tutorato 11: **matrici.**
Settimana 12. Lezioni 23 e 24: preparazione all'**esame.**
Tutorato 12: **matrici** (risoluzione di sistemi).

Appendice al Corso. Un esempio di programma: una animazione di spirali. **Non fa parte del corso**. Tutorato 13 *(di solito non viene svolto nel corso)*: materiale per la preparazione all'esame.

Settimana 01 – Output e Variabili
“1’ABC”

1. **Lezione 01: presentazione del corso, uso del main e dell’istruzione cout.**
2. **Lezione 02: dichiarazioni di variabili.**
3. **Tutorato 01: output e variabili.**



“Ascii art”

Lezione 01 - Output
Parte 1. Esempi (Cap. 1 libro di testo)

Questa lezione non include esercizi.

Lezione 01. Prima Ora. Perché un corso di Programmazione? (Presentazione del Corso, vedi Introduzione). Organizzazione del corso.

Lezione 01. Seconda Ora. Presentiamo un compilatore C++, e lo utilizziamo per scrivere un primo programma in C++.
Un ***compilatore*** è un programma che prende un file contenente programma scritto in C++ e lo trasforma in un file eseguibile. Un programma C++ è scritto in un linguaggio convenzionale ed è fatto per essere leggibile da un essere umano. Un file eseguibile, invece, è immaginato per essere letto soltanto dal computer. Nel nostro caso, i file scritti in C++ si riconoscono perche' hanno un suffisso ***.cpp***, o comunque sono etichettati "C++" in quelche modo. I file eseguibili hanno un suffisso ***.exe***, o comunque sono etichettati come "eseguibili" in qualche modo. Vediamo ora un primo programma C++, che stampa alcuni messaggi e alcuni valori numerici usando l'istruzione ***cout***. In questo corso useremo il Dev-C++, un compilatore C++ per trasformare questo programma in un eseguibile e per vederne il risultato.
Un'avvertenza sul Dev-C++. *È un compilatore vecchio e richiede che i nomi dei file .cpp non contengano spazi. Quindi scrivete "Esercizio1.cpp", e non "Esercizio 1.cpp".*

```
//Lezione01. Il comando di output cout
/* LIBRERIE. Ogni programma C++ inizia con delle librerie: sono
delle raccolte di comandi da aggiungere al C++ */

/* <math.h> Libreria di comandi per calcolare le funzioni
matematiche: pow (potenza), sqrt (radice quadrata) */
#include <math.h>
/* Libreria di comandi per stampare l'output: cout */
#include <iostream>
/* Libreria di comandi per il sistema operativo, per esempio per
windows: sistem("pause") ferma l'esecuzione del programma */
#include <stdlib.h>
using namespace std; //non spieghiamo questo comando
```

```
/* Alle librerie fa seguito un main, o "parte principale" del
programma. Il main contiene i comandi C++ che vogliamo far
eseguire dal programma */

int main ()
{cout << "STAMPA DELL'INTERO 2  : " << 2     << endl;
cout << "STAMPA DEL REALE 100.0: " << 100.0 << endl;
//Esempi delle quattro operazioni
cout << "10+10 = " << 10+10 << endl;
cout << "10-10 = " << 10-10 << endl;
cout << "10*10 = " << 10*10 << endl;
cout << "10/10 = " << 10/10 << endl;
//Esempi di funzioni di libreria
cout << "10 elevato a 10 = " << pow(10,10) << endl;
cout << "10 elevato a  2 = " << pow(10,2)  << endl;
/* Un esempio di overflow: un valore al di la' dei valori
rappresentabili viene trattato come infinito */
cout << "10 elevato a 1000    = " << pow(10,1000) << endl;
//Esempi di errori: 1/0 fa saltare il programma
//cout << "1/0              = " << 1/0 << endl;
cout << "sqrt(100) = " << sqrt(100) << endl;
//Esempio di un valore nan (NOT A NUMBER):non è un numero (reale)
cout << "sqrt(-1)  = " << sqrt(-1) << endl;
/* L'arrotondamento puo' far si' che la divisione tra interi dia
zero */
cout << "1/10      = " << 1/10   << endl; //otteniamo 0
//La divisione tra reali non ha arrotondamento
cout << "1./10.    = " << 1./10. << endl;
/* è possibile forzare l'intero 10 a essere un reale con:(double)
10. Tuttavia, stampando non notiamo differenze tra 10 intero e 10
reale */
cout << "(double) 10 = " << (double) 10 << endl;
/* Comunque le differenze ci sono: se calcoliamo 1/(double) 10
otteniamo 0.1, un numero reale, e non 0 */
cout << "1/(double) 10  = " << 1/(double) 10   << endl;

system("pause");
}

/* system("pause") sospende il programma e ci fa vedere i
risultati ottenuti. Senza di essa, il Dev-C++ chiude il programma
quando arriva alla fine, e i risultati spariscono. system("pause")
si può usare solo con Windows. Se usate un altro sistema
operativo, aggiungete la libreria #include <stdio.h>, quindi
rimpiazzate system("pause") con getchar();*/
```

Lezione 02 - Variabili
Parte 1. Esempi (Cap. 2 libro di testo)
"Il contenuto di una scatola"

Prima ora: cap. 2 libro di testo.

Variabili. Nella prima ora di lezione presentiamo la nozione di "tipo" e di "variabile" in C++. Un tipo indica la rappresentazione in C++ di un insieme di un insieme di oggetti. Oggi vedremo i tipi ***int*** (interi da -2^{31} fino a $+2^{31}$), ***double*** (numeri reali con finite cifre decimali, da -10^{300} a $+10^{300}$ circa), e ***char*** (l'insieme dei 256 caratteri tipografici dell'alfabeto ASCII). Una variabile di un tipo int indica un oggetto di tipo int in C++, e così per i tipi double, char, eccetera.

Cosa sono le variabili in programmazione? In programmazione, una "variabile" x ***non è una incognita***, va invece pensata come una "scatola" con dentro un valore: noi dobbiamo scegliere il contenuto iniziale di questa "scatola", e possiamo modificarlo in ogni momento. ***È un errore concettuale usare una variabile prima di assegnarle un valore, come faremmo con una variabile in Matematica.*** Nel C++, se non scegliamo il valore di una variabile, lo sceglie il compilatore al posto nostro, sostanzialmente a caso, e il comportamento del programma diventa imprevedibile.

Cosa servono le variabili? Una variabile ci serve a dare un nome e quindi una spiegazione a un valore o una formula: in ogni programma, dobbiamo rimpiazzare il più possibile valori e espressioni con variabili per spiegare cosa stiamo facendo. Inoltre un valore assegnato a una variabile è facile da ritrovare, facile da modificare, e non deve essere ricalcolato.

```
// Lezione02-variabili
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

// ORA 1. un main che dichiara alcune variabili
int main ()
{/* e' un errore concettuale usare una variabile prima di
assegnarle un valore: si ottiene un risultato non prevedibile. */
 int a; cout << "a = " << a << endl;
 // il valore della variabile a stampato e' imprevedibile

 //Ora dichiariamo e assegnamo una variabile di tipo intero: int
 int x = 123; // possibile perche' 123 è un numero intero
 cout << "x = " << x << endl; //stampa 123

 //possiamo riassegnare il valore di x:
 //in questo caso NON dobbiamo ripetere il tipo di x
 x = 124; //riassegno a 124 e NON ne ripeto il tipo
 cout << "x = " << x << endl; //stampo 124

//dichiariamo una variabile di tipo reale: double
 double y = 123.4; //possibile perche' 123.4 è un numero reale
 cout << "y = " << y << endl;
 y = 124.5;
 cout << "y = " << y << endl;
//Anche un carattere, se posto tra apici, è un valore
 cout << "} = " << '}' << endl;

//possiamo dichiarare variabili di tipo carattere
char carattere = 'a';
cout << "carattere=" << carattere << endl;
carattere = '\212'; //Il nome convenzionale del carattere 'è'
cout << "carattere=" << carattere << endl;

/* Lo stesso oggetto 5 puo' essere visto come un messaggio da
stampare */
cout << "5" << endl;
//oppure come carattere
cout << '5' << endl;
//oppure come valore intero
cout <<  5  << endl;
//oppure come valore reale
cout <<  5.  << endl;
```

```
//alla stampa non c'e' nessuna differenza
//invece per il C++ sono oggetti distinti.

//un esempio di assegnazione e stampa di piu' tipi
int hour, minute;
char colon;
hour = 11;
minute = 59;
colon = ':';
cout << "The current time is "
<< hour << colon << minute << endl;

//Attenti alla precedenza degli operatori
cout << "2*3-1 vale " << 2*3-1
<< " mentre 2*(3-1) vale " << 2*(3-1) << endl;

/* La precedenza degli operatori causa un arrotondamento oppure un
altro. Questo puo' notevolmente influire sul risultato */
cout << "A causa degli arrotondamenti (59*100)/60 vale "
<< endl << (59*100)/60 << endl;
cout << "Invece 59*(100/60) vale "
<< endl << 59*(100/60) << endl;

//Qualche esempio di operazioni sui caratteri
char letter;
letter = 'a' + 1; //stampa b, il successore di a
cout << letter << endl;
/* Stampa del numero del carattere nell'alfabeto del computer
(detto ASCII) */
cout << "numero del carattere = "
<< (int) letter << endl;
//(int) letter prende il carattere dentro letter
//e lo rimpiazza con la sua posizione nell'alfabeto ASCII

system("pause");
}
```

Lezione 02 - Variabili
Parte 2. Esercizi

Scrivete un programma che dichiara variabili per contenere base e altezza di un rettangolo e raggio di un cerchio. Il programma quindi calcola e stampa sullo schermo:

2.1. base, altezza, area, perimetro e diagonale del rettangolo;
2.2. raggio, area, diametro e circonferenza del cerchio.

Suggerimenti.

- Trattandosi di un problema di geometria, conviene usare variabili di tipo numero reale, perché dati e risultati sono numeri reali.

- In un programma, la leggibilità è essenziale perché consente di trovare e correggere gli errori. Per migliorare la leggibilità, vi chiediamo di scegliere per ogni variabile un nome che ne spiega l'uso (base, altezza, eccetera), e di assegnare i risultati a variabili prima di stamparli. Questo accorgimento rende chiaro quale valore viene stampato da un cout.

- Evitate il più possibile di usare costanti nei vostri programmi, assegnate i valori a variabili e usate le variabili. In questo modo diventa facile modificare un programma: basta cambiare ogni costante una volta sola, quando viene assegnata a una variabile.

- *Ricordate, il **Dev-C++.** è un compilatore vecchio e richiede che i nomi dei file .cpp non contengano spazi. Quindi scrivete "Esercizio1.cpp", e non "Esercizio 1.cpp".*

Lezione 02 - Variabili
Parte 3. Soluzioni

```
// Lezione02-variabili-Soluzioni
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

int main ()
{double base                  = 5.0, altezza = 6.0;
 double area                  = base*altezza;
 double diagonale             = sqrt(base*base+altezza*altezza);
 double perimetro             = 2*(base+altezza);

/* Es. 2.1 RETTANGOLO. Per leggibilità, assegnamo i valori a
variabili prima di stamparli */
 cout << endl << "RETTANGOLO" << endl;
 cout << "base rettangolo    = " << base      << endl;
 cout << "altezza rettangolo = " << altezza   << endl;
 cout << "area               = " << area      << endl;
 cout << "diagonale          = " << diagonale << endl;
 cout << "perimetro          = " << perimetro << endl;

/* Es. 2.2 CERCHIO. Per leggibilità, assegnamo i valori a
variabili prima di stamparli */
 cout << endl << "CERCHIO" << endl;
 double raggio                = 5.0;
 double pigreco               = 3.14159265;
 double cerchio               = raggio*raggio*pigreco;
 double circonferenza         = raggio*2*pigreco;
 double diametro              = 2*raggio;

 cout << "raggio cerchio     = " << raggio        << endl;
 cout << "area cerchio       = " << cerchio       << endl;
 cout << "circonferenza      = " << circonferenza << endl;
 cout << "diametro           = " << diametro      << endl;

 cout << endl; system("pause");
}
```

Tutorato 01 - Output e Variabili
Parte 1. Esercizi proposti

Vi chiediamo di scrivere il più possibile gli esercizi in un solo file, dunque con un unico ***main***, con un gruppo di istruzioni per ogni esercizio. Questo semplifica il lavoro dell'esercitatore quando fate vedere il vostro programma.

In questo e nei tutorati futuri, vi chiediamo di ***concentrarvi sui primi esercizi***, di cercare di fare degli esercizi bene piuttosto che di farne tanti, e di non guardare le soluzioni prima di averle cercate voi stessi.

Tut01.1 Scrivete un programma che stampa a video: 'abbcccdddd'.

Tut01.2 Scrivete un programma che stampa a video:

a
bb
ccc
dddd

Tut01.3 Scrivete un programma che dichiara una variabile di tipo **char** e una variabile intera. Assegnate dei valori alle variabili a vostra scelta. Stampate il carattere successivo e l'intero precedente. Per esempio, assegnate alle variabili i valori b e 3 la stampa visualizza c,2.

Tut01.4 Dichiarate due variabili intere,assegnate loro dei valori a vostri scelta e calcolatene e visualizzatene la somma, la sottrazione (del primo per il secondo), il prodotto, la divisione (del primo per il secondo), il modulo (del primo rispetto al secondo). Richiamate la funzione sqrt su una delle variabili dichiarate e visualizzatene il risultato. Richiamate la funzione pow usando le variabili dichiarate. Richimate la funzione log usando come argomento una delle variabili dichiarate.

4.a provate a richiamare sqrt passando come parametro in input -1. Cosa vedete stampato a video?

4.b provate a richiamare la funzione pow usando come esponente una variabile che contenga il valore 1000. Cosa vedete stampato a video?

4.c provate a richiamare la funzione log usando come argomento il numero 0 e, successivamente, il numero 1. Cosa compare nella finestra di output?

Tut01.5 Scrivete un programma che calcoli perimetro e area di un triangolo di cui si conoscono le misure dei lati (per l'area usate la formula di Erone, che trovate su Wikipedia).

Tut01.6 Scrivete un programma che scomponga un intero non negativo in unita', decine, centinaia e migliaia. Per esempio ***123456789***

diventa ***123456*** migliaia, ***7*** centinaia, ***8*** decine e ***9*** unita'. Notate che le migliaia possono essere un intero non negativo qualunque, mentre centinaia, decine e unita' sono singole cifre. Ricordatevi che la divisione arrotondata per difetto di x per 10, 100, 1000 si indica con x/10, x/100, x/1000.

Qualche suggerimento per cominciare.
1. Chiamate sempre le variabili con nomi "evocativi" che ne identifichino il ruolo: area, perimetro, raggio, diametro eccetera. Poi evitate di chiamare le variabili allo stesso modo del loro contenuto, confonde le idee. Per es.

```
char b = 'b';            // è da evitare
char lettera1 = 'a';     // è ok
```

2. Evitate di usare una sequenza di caratteri dove è richiesto un singolo carattere e viceversa. Un esempio: se assegnate una sequenza di caratteri ad una variabile x di tipo char, questa viene troncata. Non potete scrivere:

```
char x = "abc";          // x contiene soltanto c
```

Un altro esempio. Non potete neppure scrivere 'abcd', perche' l'apice singolo si usa solo per indicare singoli caratteri. Se scrivere ***cout << 'abcd'*** utilizzando apici singoli '' a video verrà stampato un numero senza senso (e' la codifica numerica della sequenza di caratteri)! Per stampare abcd dovete usare un apice doppio: ***cout << "abc"***.

Infine ricordate che il ***Dev-C++.*** *è un compilatore vecchio e richiede che i nomi dei file .cpp non contengano spazi. Quindi scrivete "Esercizio1.cpp", e non "Esercizio 1.cpp".*

Tutorato 01 - Output e Variabili
Parte 2. Soluzioni

Nota. Per aiutarvi al leggere le soluzioni noi abbiamo separato ogni soluzioni in un main diverso. A voi chiediamo invece di scrivere il piu' possibile gli esercizi in un solo file, dunque con un unico ***main***, con un gruppo di istruzioni per ogni esercizio. Questo ci aiuta nella correzione.

Tut01.1. Scrivere un programma che stampa a video: 'abbcccdddd'.

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
using namespace std;

int main()
{
    cout << "abbcccdddd" << endl;
    system("pause");
}
```

Tut01.2. Scrivere un programma che stampa a video:

```
a
bb
ccc
dddd
```

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
using namespace std;

int main()
{
    cout << "a" << endl;
    cout << "bb" << endl;
    cout << "ccc" << endl;
    cout << "dddd" << endl;
    system("pause");
}
```

Tut01.3. Scrivete un programma che assegna un carattere e un intero e stampa il carattere successivo e l'intero precedente. Per esempio, se assegnate b,3 la stampa visualizza c,2.

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
using namespace std;
```

```
int main()
{
    char c = 'b';
    int num = 3;
    c = c + 1;
    num = num - 1;
    cout << c << endl;
    cout << num << endl; system("pause");}
```

Tut01.4. Dati 2 numeri interi in assegnazione, calcolarne e visualizzarne la somma, la sottrazione (del primo per il secondo), il prodotto, la divisione (del primo per il secondo) ed il modulo (del primo rispetto al secondo).

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
using namespace std;
/*
calcola somma, sottrazione, moltiplicazione, divisione e modulo di
due numeri forniti in input
*/

int main()
{
 int a=5, b=7;
 cout << " a = " << a << " b = " << b << endl;
cout << "Calcolo somma, sottrazione, moltiplicazione, divisione e
modulo di due interi forniti in assegnazione" << endl;
 cout << "Somma: " << a << " + " << b << " = " << a+b << endl;
 cout << "Sottraz.: " << a << " - " << b << " = " << a-b << endl;
 cout << "Prodotto: " << a << " * " << b << " = " << a*b << endl;
 cout << "Divisione: " << a << " / " << b << " = " << a/b << endl;
 cout << "Modulo: " << a << " mod " << b << " = " << a%b << endl;
 system("pause");}
```

Tut01.5. Scrivere un programma che calcoli perimetro e area di un triangolo di cui si conoscono le misure dei lati (per l'area usate la formula di Erone).

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
using namespace std;
int main () {
double a=3, b=5, c=7, perimetro, sp, area;
cout << "Lunghezza dei 3 lati del triangolo: " << '\n';
cout << " a = " << a << " b = " << b << " c = " << c << endl;
perimetro = a+b+c;
sp = perimetro/2;
area = sqrt(sp *(sp-a)*(sp-b)*(sp-c));
```

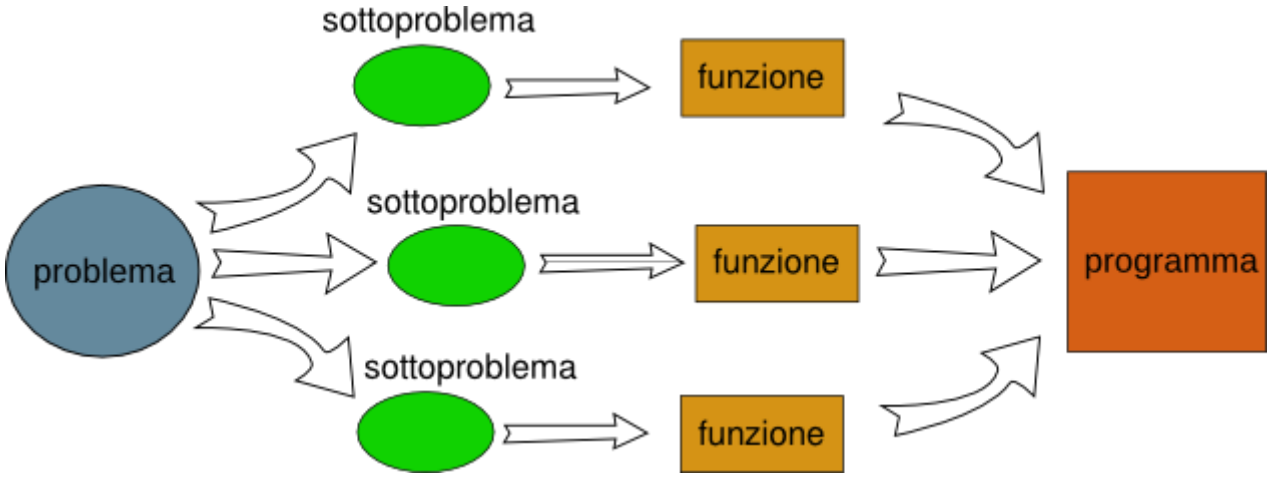
```
cout << "Il perimetro del triangolo vale: " << perimetro << " e
l'area vale: " << area << '\n';
system("PAUSE");
}
```

Tut01.6. Scrivete un programma che scomponga un intero non negativo in unita', decine, centinaia e migliaia. Per esempio ***123456789*** diventa ***123456*** migliaia, ***7*** centinaia, ***8*** decine e ***9*** unita'. Notate che le migliaia possono essere un intero non negativo qualunque, mentre centinaia, decine e unita' sono singole cifre. Ricordatevi che la divisione arrotondata per difetto di x per 10, 100, 1000 si indica con x/10, x/100, x/1000.

```
#include <iostream>
#include <stdlib.h>
#include <math.h>
using namespace std;

int main (){int numero = 1234; //scegliamo quale numero scomporre
cout << "Un numero intero non negativo " << endl;
cout << "(che sia piu piccolo di 10.000)"<< endl;
cout << numero << endl;
int migliaia, centinaia, decine, unita;
migliaia = numero/1000;
centinaia = (numero/100)-(migliaia*10);
decine = (numero/10)-(migliaia*100+centinaia*10);
unita = numero-(migliaia*1000+centinaia*100+decine*10);
cout << "Il numero e\' composto da: "<< '\n';
cout << migliaia << " migliaia, " << centinaia;
cout << " centinaia, " << decine << " decine, " << unita << "
unita."<< endl;
system("PAUSE"); }
```

Settimana 02 – Funzioni e Condizionale
"Come scomporre un problema"

1. **Lezione 03: definizione di funzioni in informatica.**
2. **Lezione 04: condizionale e istruzione return;**
3. **Tutorato 02: funzioni e condizionale.**



In informatica, "funzione" significa "sottoprogramma" e si usa per scomporre un problema in problemi più piccoli

Lezione 03 - Funzioni
Parte 1. Esempi (Cap. 3 libro di testo)

Prima ora. Lezione 03. Una funzione è una parte di un programma dedicato alla soluzione di un sotto-problema. Le funzioni vengono usate per semplificare il lavoro di ***scrivere*** e soprattutto di ***correggere*** un programma. Quando scriviamo una funzione, anziché affrontare il problema originario affrontiamo un problema più semplice, e anche la correzione risulta facilitata, perché gli errori vanno cercati tra poche istruzioni. Alla fine il problema originario viene risolto usando tutte le funzioni che abbiamo definito. Un altro vantaggio delle funzioni è che ci ***consentono di riutilizzare facilmente*** e di ***modificare facilmente*** lunghi gruppi di istruzioni.

Questa settimana vediamo come definire le funzioni dette "**VOID**", che sono le funzioni più semplici del C++. Per cominciare, vediamo le regole che ci consentono di definire e usare funzioni che stampano qualche cosa sullo schermo. Usare una funzione si dice: ***"chiamare una funzione"***. Il main stesso è una funzione.

```
// Lezione03-funzioni VOID
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

/* ORA 1. Funzioni VOID. Il comando per definire nuove funzioni
VOID è
                VOID NAME ( LIST OF PARAMETERS )
                        {STATEMENTS}
La funzione definita è un comando che modifica la memoria oppure
lo schermo. La lista () di parametri puo' essere vuota. La
definizione deve avvenire prima del main e fuori da altre
funzioni. Le variabili dichiarate in una funzione sono considerate
distinte da quelle dichiarate in una qualunque altra funzione
oppure nel main, anche quando il nome è lo stesso.

Definiamo una nuova funzione VOID che <<va a capo>>, con 0
parametri, poi vedremo come usarla. */
void newLine()
 {cout << endl;}

//definiamo una funzione VOID usando una precedente funzione
```

```
void threeLine ()
{/* Quando "chiamiamo" (usiamo) una funzione definita come "VOID
f(tipo1 x1, ...){...}" dobbiamo rispettare le seguenti regole:
  1.non ripetiamo VOID, tipo1, ... e la definizione {...}
  2.scriviamo sempre f(...), anche quando f non ha parametri */
newLine (); newLine (); newLine ();}

//esempio di funzione VOID con 1 parametro: stampa 2 volte un
carattere
void printTwice (char phil)
{cout << phil << phil << endl;}

//esempio di funzione VOID con 2 parametri: stampa di una data
void printTime (int hour, int minute)
{cout << hour; cout << ":"; cout << minute;}

/* Esempi di ERRORI: ci sono alcuni errori tipici da evitare
quando "chiamiamo" una funzione. Tutti gli errori che vedete
elencati qui sotto bloccano il programma già nella fase di
compilazione */
// void newLine() {cout << endl;} // Non si possono definire
funzioni dentro il main
// newLine;  //il solo nome della funzione non basta per eseguire
la funzione
// cout << newLine(); //newLine è un sottoprogramma, non è un
valore, quindi non puo' essere stampato
// cout << newLine()+7; //non posso sommare 7 a un valore
inesistente

// Notate che il main è una funzione con lista di parametri vuota
int main ()
{//"chiamate di funzioni" con 0 parametri
 cout << "Sperimento 3 volte il comando newLine:";
 newLine();newLine();newLine(); //newLine si comporta come un
comando del C+
 cout << "Sperimento il comando threeLine:";
 threeLine(); //threeLine si comporta come 3 volte newLine

 //"chiamate di funzioni" con 1 parametro
 cout << "Sperimento printTwice('X'):";
 //quando "chiamo" printTwice NON RIPETO il tipo char di 'X'
 printTwice('X');

 cout << "Sperimento printTwice(argument):";
 //quando "chiamo" printTwice NON RIPETO il tipo char di argument
```

```
char argument = 'a'; printTwice(argument);
cout << "Sperimento di nuovo printTwice(argument):";
argument = 'b'; printTwice(argument);

//"chiamate di funzioni" con 2 parametri
cout << "Sperimento printTime(11,59);" << endl;
printTime(11,59); newLine(); system("pause");}
```

Lezione 03 - Funzioni void
Parte 2. Esercizi

Seconda ora. Lezione 03. Nella seconda ora dovete scrivere voi stessi delle "funzioni C++" e "chiamarle" (cioe' usarle) in un programma C++. Dovete riprendere e modificare l'ultimo programma visto nel Tutorato 01, che scomponeva un numero nelle sue cifre, e trasformarlo da programma come era in una funzione che un programma puo' chiamare dal main.

3.1. scrivete una funzione VOID:
void decomponiInt(**int** x){...}
che stampa le cifre delle centinaia, decine e unita' dell'intero x>=0. **Suggerimento.** Le cifre richieste si possono ottenere usando la divisione intera y/100, y/1000, … e la funzione y%10, y%100, ... che calcola il resto della divisione intera di y per 10, per 100, ..

3.2. modificate la funzione VOID dell'esercizio 3.1 come segue:
void decomponiDouble(**double** x){...}
per far stampare le cifre delle centinaia, decine e unita' e la PARTE DECIMALE del reale x>=0. **Suggerimento.** Rispetto all'esercizio precedente, non potete usare direttamente la divisione intera e l'operazione % di resto della divisione intera su un numero reale. Usate prima la funzione (**int**) y per rimpiazzare un reale y con la usa parte intera.

3.3. Scrivete un **main** in cui sperimentate ***molte volte*** ciascuna funzione.

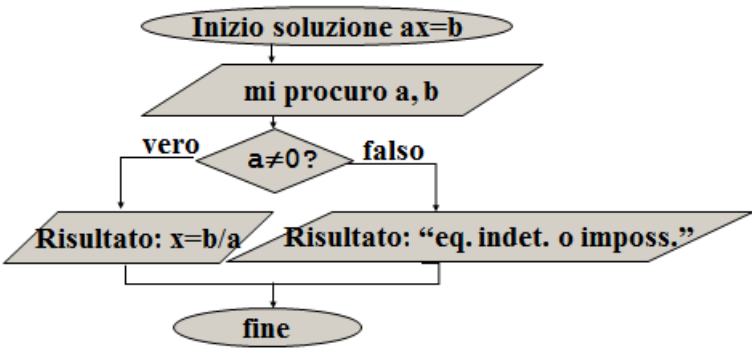
Nota. Scomporre un programma in funzioni è utile a ridurre gli errori solo se controlliamo accuratamente una funzione dopo averla definita. Prima di usare una funzione in un programma dobbiamo essere sicuri che funzioni bene. Altrimenti scomporre un programma in funzioni **diventa inutile**: gli errori in una funzione si ripercuotono in errori in ogni programma che usa quella funzione, errori la cui origine diventa molto difficile da rintracciare perche' sono stati fatti tempo prima e in un'altra parte del programma.

Riassumendo: il lavoro di un programmatore ***non termina dopo aver definito una funzione***: è essenziale controllare il buon funzionamento di una funzione, innanzitutto su esempi, e poi ragionandoci su.

Lezione 03 - Funzioni void
Parte 3. Soluzioni

```
// Lezione03-esercizi su funzioni VOID - Soluzioni
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

/* 3.1 Decomposizione di un intero */
void decomponiInt(int x)
{
 cout << "cifra centinaia x = " << (x%1000)/100 << endl;
 cout << "cifra decine    x = " << (x%100)/10   << endl;
 cout << "cifra unita'    x = " << (x%10)       << endl;
}

/* 3.2 Decomposizione di un reale */
void decomponiDouble(double y)
{
 decomponiInt((int) y);
 cout << "parte decimale  y = " << y-((int) y) << endl;
}

/* 3.3 Esempi di uso delle funzioni precedenti
 È essenziale controllare una funzione dopo averla definita. */
int main ()
{int x;
 cout << "   1.Test decomponiInt"   << endl;
 x=0;cout << "x=" << x << endl; decomponiInt(x); cout << endl;
 x=1111;cout << "x=" << x << endl; decomponiInt(x); cout << endl;
 x=1234;cout << "x=" << x << endl; decomponiInt(x); cout << endl;
 x=34;cout << "x=" << x << endl; decomponiInt(x); cout << endl;

 double y;
 cout << "   2.Test decomponiDouble"<< endl;
 y=0.;cout << "y=" << y << endl; decomponiDouble(y); cout << endl;
 y=1111.1;cout << "y=" << y << endl;decomponiDouble(y);cout<<endl;
 y=1234.5;cout<<"y="<< y << endl;decomponiDouble(y);cout<< endl;
 y=34.5;cout << "y="<< y << endl;decomponiDouble(y);cout << endl;

 system("pause");}
```

Lezione 04 - Condizionale
Parte 1. Esempi (Cap. 4 libro di testo)
"Un'istruzione definita per casi"



I calcoli svolti per risolvere l'equazione ax=b
cambiano a seconda del valore di a

Lezione 04. In questa lezione vedremo l'istruzione condizionale ***if***, che consente di eseguire oppure no un gruppo di istruzioni a seconda della verita' di una condizione. Vedremo tre tipi di ***if***.

1. L'***if*** tronco:

```
if (condizione)
 {istruzione1; istruzione2; …}
/* se la condizione è vera viene eseguito il gruppo di istruzioni,
altrimenti l'if fa nulla */
```

2. L'***if*** a due rami:

```
if (condizione)
 {istruzione1; istruzione2; …}
else
  {istruzione1; istruzione2; …}
/* se la condizione è vera viene eseguito il gruppo di istruzioni,
altrimenti viene eseguito il secondo gruppo */
```

3. La catena di ***if***:

```
if (condizione1)
 {istruzione1; istruzione2; …}
else if (condizione2)
  {istruzione1; istruzione2; …}
else
 {istruzione1; istruzione2; …}
```

```
/* se la prima condizione è vera viene eseguito il gruppo di
istruzioni, se la seconda condizione è vera viene eseguito il
secondo gruppo di istruzioni, altrimenti viene eseguito l'ultimo
gruppo di istruzioni */

// Lezione04-condizionale, return senza un valore
//ORA 1. condizionale e istruzione return;
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

//questa funzione stampa se x è pari oppure dispari
void printParity (int x); //rimandiamo la definizione

/* questa funzione stampa log(x) se x>0, altrimenti avvisa che x è
<=0 e dunque il logaritmo non puo' venire calcolato */
void printLogarithm (double x); //rimandiamo la definizione

int main ()
{ int x=3, y=5;
 cout << " ESEMPI DI IF TRONCO" << endl;
//se vale una condizione si svolge una azione
//altrimenti si fa nulla
 if (x > 0)
 {cout << " valore " << x << " positivo" << endl;}

/* Gli enunciati per cui esiste un metodo di calcolo hanno un
valore di verita' o "booleano" 0=falso, 1=vero. L'uguaglianza di
rappresenta con un doppio uguale mentre l'uguale indica
l'assegnazione. L'insieme {0,1} dei "booleani" si indica con bool.
*/
 cout << " ESEMPI DI CONDIZIONI" << endl;
 cout<<" x=" << x << " y=" << y << endl;
 cout<<"valore (x == y) "<<(x == y)<<endl;// x uguale a y
 cout<<"valore (x != y) "<<(x != y)<<endl;// x diverso da y
 cout<<"valore (x  > y) "<<(x  > y)<<endl;// x maggiore di y
 cout<<"valore (x  < y) "<<(x  < y)<<endl;// x minore di y
 cout<<"valore (x >= y) "<<(x >= y)<<endl;// x magg. o uguale di y
 cout<<"valore (x <= y) "<<(x <= y)<<endl;//x minore o uguale di y

 cout << " ESEMPI DI IF CON DUE POSSIBILITA'" << endl;
//x%2 è il resto della divisione di x per due, dunque x%2==0
significa x pari
 if (x%2 == 0) //x vale 5, dunque x%2 vale 1, non 0
```

```
  {cout << "valore " << x << " pari" << endl;}
 else //dato che x%2 vale 0, viene eseguito questo lato
  {cout << "valore " << x << " dispari" << endl;}

//un esempio usando una funzione
 cout << " UNA FUNZIONE CHE USA IF" << endl;
 int number = 17;
 printParity (number); /* non scrivete invece: printParity (int
number); perche' il tipo si usa solo nelle definizioni */

/*Si possono scrivere condizionali dentro condizionali, in due
modi.
 1. condizionali a catena: inseriro un condizionale dentro la
parte "else" di un altro condizionale. Condizionali a catena
descrivono una lista di scelte, dove ogni scelta richiede aver
scartato le precedenti. */

 cout << " CONDIZIONALI A CATENA" << endl;
 if (x > 0)
 {cout << "valore " << x << " positivo" << endl;}
 else if (x < 0)
 {cout << "valore " << x << " negativo" << endl;}
 else
 {cout << "valore " << x << " nullo" << endl;}

/*2. condizionali dentro condizionali: sono piu' difficili da
leggere quindi vanno evitati quando possibile */
 cout << " CONDIZIONALI DENTRO CONDIZIONALI" << endl;
 if (x == 0)
  {cout << "valore " << x << " nullo" << endl;}
 else //tutto cio' che segue sta dentro l'ELSE di un condizionale
      //all'interno dell'ELSE x non è nullo
  {if (x > 0)
   {cout << " valore " << x << " positivo" << endl;}
  else //x non è nullo, quindi se non è positivo è negativo
   {cout << " valore " << x << " negativo" << endl;}
  }

//esempi di funzione con una stampa definita per casi
 cout << " UNA FUNZIONE CON MESSAGGIO DI ERRORE" << endl;
 cout << "logaritmo di 0:" << endl;
 printLogarithm(0); //viene stampato un messaggio di errore
 cout << "logaritmo di 10:" << endl;
 printLogarithm(10); //viene stampato il logaritmo di 10
```

```
/* a questo punto il libro di testo inserisce un cenno alla ricorsione, che pero' noi vedremo solo tra qualche lezione */
 cout << endl; system("pause");
}

//questa funzione stampa se x è pari oppure dispari
void printParity (int x) {
if (x%2 == 0)
 {cout << "valore " << x<< " pari" << endl;}
else
 {cout << "valore " << x<< " dispari" << endl;}
}

/* questa funzione stampa log(x) se x>0, altrimenti avvisa che x è <=0 e dunque il logaritmo non puo' venire calcolato */
void printLogarithm (double x)
{if (x <= 0.0)
 {cout << "valore " << x << " non positivo" << endl;
  return;}  /*l'istruzione return; termina l'esecuzione della funzione, evitando di eseguire le righe successive e quindi di calcolare il logaritmo */
 double result = log (x);
 cout << "Il logaritmo di " << x << " vale " << result;
}
```

Lezione 04 - Condizionale
Parte 2. Esercizi

Seconda ora. Esempi di condizionale e istruzione **return.**
Nell'ultimo esercizio introduciamo l'istruzione cin >> x; che ferma l'esecuzione del programma, aspetta che noi introduciamo una valore per x, lo assegna a x e riprende l'esecuzione del programma.

4.1 Scrivete una funzione

void risolvi(**double** a, **double** b)

che risolva una equazione di primo grado ax=b con la formula b/a, senza usare un **if.**

4.2. Scrivete una seconda funzione

void risolvi2(**double** a, **double** b)

che risolva una equazione di primo grado a*x=b, distinguendo i casi: equazione determinata con soluzione b/a, equazione indeterminata, equazione impossibile, usando una catena di **if.**

4.3. Scrivete una funzione

void stamparadice(){…}

che non usa argomenti. stamparadice usa l'istruzione ***cin >> x;*** chiede all'utente del programma un valore x. Quindi se x è positivo o nullo ne stampa la radice quadrata, se è negativo stampa un messaggio: "non esiste la radice quadrata di un numero negativo".

4.4. Scrivete un **main** in cui sperimentate ***molte volte*** ciascuna funzione. Ricordatevi: è essenziale controllare le funzioni subito dopo averle definite, per evitare che l'uso di funzioni che contengano errori produca in futuro errori di cui è difficile capire la provenienza.

- Per controllare risolvi(a,b) e risolvi2(a,b) basta "chiamare" queste funzioni su molti valori fissi, per esempio con: risolvi(1,2), risolvi(0,2), risolvi2(1,2), risolvi2(0,2), eccetera.
- Per controllare stamparadice() dovete invece "chiamarlo" nel main, ma senza argomenti (perche' non ne ha). Durante l'esecuzione del programma, tocchera' a voi inserire un valore di x da tastiera su cui sperimentare la funzione.

Lezione 04 - Condizionale
Parte 3. Soluzioni

```
// Lezione04-condizionale-Soluzioni
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

/* 4.1 soluzione equazione primo grado */
void risolvi(double a, double b)
{
 cout << "equazione " << a << "*x = " << b << endl;
 cout << " soluzione x = " << b/a << endl;
}

/* 4.2 discussione equazione primo grado */
void risolvi2(double a, double b)
{
 cout << "equazione " << a << "*x = " << b << endl;
 if (a !=0)
  {cout << " eq. determinata soluzione x = " << b/a << endl;}
 else if (b==0) //a=0
  {cout << " eq. indeterminata" << endl;}
 else //b=/=0 e a=0
  {cout << " eq. impossibile" << endl;}
}

/* 4.3 richiesta di un x e stampa di sqrt(x) se x>=0 */
void stamparadice()
//questa funzione utilizza cin per chiedere un valore
{double x; //innanzitutto dobbiamo dichiarare x
 cout << "Inserite un valore per x" << endl;
 cin  >> x;
 if (x >=0)
  {cout << "Radice di x = " << sqrt(x) << endl;}
 else
  {cout << "Non esiste la radice di un x negativo" << endl;}
}

/* 4.4 esempi di uso di funzioni precedenti */
int main (){cout << endl << "ESEMPI USO FUNZIONE RISOLVI" << endl;
 risolvi(2,0); //risolve 2*x=0
 risolvi(2,1); //risolve 2*x=1
```

```
 risolvi(0,1); //tenta di risolvere 0*x=1 e produce inf=infinito
 risolvi(0,0);  //tenta  di  risolvere  0*x=0  e  produce  nan=not  a
number

 cout << endl << "ESEMPI USO FUNZIONE RISOLVI2" << endl;
 risolvi2(2,0); //risolve 2*x=0 con il valore x=0
 risolvi2(2,1); //risolve 2*x=1 con il valore x=0.5
 risolvi2(0,1); //risolve 0*x=1 con un messaggio "indeterminata"
 risolvi2(0,0); //risolve 0*x=0 con un messaggio "impossibile"

 cout << endl << "ESEMPI USO FUNZIONE STAMPARADICE" << endl;
 stamparadice(); //tocca a noi fornire un x da tastiera
 stamparadice(); //tocca a noi fornire un x da tastiera
 cout << endl; system("pause");}
```

Tutorato 02 – funzioni e condizionale
Parte 1. Esercizi proposti

Nota. Per aiutarvi al leggere le soluzioni di questi esercizi, troverete ogni soluzione in un main diverso. A voi invece chiediamo di scrivere il piu' possibile gli esercizi in un solo file, dunque con un unico ***main***, con un gruppo di istruzioni per ogni esercizio. Questo semplifica il lavoro di tutorato.

Ricordatevi di ***concentrarvi sui primi esercizi***.

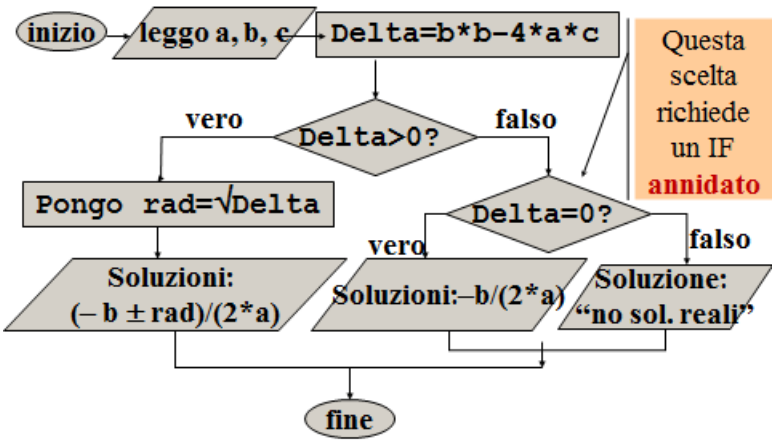
Istruzione di Input. In questi esercizi utilizziamo l'istruzione ***cin >> x;*** Questa istruzione ferma l'esecuzione del programma, aspetta che noi introduciamo una valore per x, lo assegna a x e riprende l'esecuzione del programma.

Tut02.1. Scrivete un programma che, ricevuta in input una lingua dall'utente (italiano, inglese, francese, tedesco o spagnolo), lo saluti nella lingua scelta ("Ciao", "Hello", "Salut", "Tschuss" o "Hola"). **Avvertenza.** Per scegliere la lingua l'utente dovra' inserire una lettera tra i,e,f,d,e, che voi dovrete assegnare con un ***cin*** a una variabile di nome lingua e di tipo ***char***. Aggiungete subito dopo una istruzione: ***lingua = tolower(lingua);*** che rende la lettera inserita minuscola, altrimenti se l'utente scrive "I" al posto di "i" non viene capito.

Tut02.2. Scrivete un programma che, presi in input due punti del piano distinti, restituisca la loro distanza euclidea. Per questo esercizio vi chiediamo di usare una funzione.

Tut02.3. Scrivere un programma che, usando le funzioni, calcoli le soluzioni di un'equazione di secondo grado i cui coefficienti sono dati in input:

a. richiedendo all'utente di inserire parametri per cui l'equazione abbia due soluzioni reali distinte;
b. considerando anche i casi delle soluzioni coincidenti e immaginarie.



I calcoli svolti per risolvere l'equazione $ax^2+bx+c=0$ cambiano a seconda dei valori di a,b,c

Tut02.4 (Attenzione: questo esercizio richiede di distinguere tra molti casi. I primi 3 esercizi sono più che sufficienti!).
Scrivete un programma che, inseriti i numeri (a1, b1, c1, a2, b2, c2), coefficienti di due rette a1*x + b1*y + c = 0, a2*x + b2*y + c = 0, dica se (a1, b1, c1, a2, b2, c2) sono i coefficienti di due rette, e se queste rette sono parallele, perpendicolari, incidenti o coincidenti. (Usate le funzioni).

Suggerimento. Distinguete 4 casi: b1=0 e b2=0, b1=0 e b2≠0, b1≠0 e b1=0, b1≠0 e b2≠0. Nei primi tre casi distinguete dei sottocasi a seconda dei valori di a1,a2,c1,c2. Nel quarto caso trasformate le rette nella forma y = m1*x+q1 e y = m2*x+q2, dove m1=-(a1/b1), q1=-(c1/a1), m2=-(a2/b2) e q2 = -(c2/a2). Se m1*m2=-1 le due rette sono ortogonali, se m1=m2 e q1=q2 sono coincidenti, se m1=m2 e q1≠q2 sono parallele, altrimenti sono incidenti. Usate un if "annidato" per separare i vari sotto-casi.

Tutorato 02 – funzioni e condizionale
Parte 2. Soluzioni

Nota. Ricordatevi di scrivere le soluzioni in un unico ***main***. Noi invece, per facilitarvi la lettura, le scriviamo in main distinti.

Tut02.1 Scrivere un programma che, ricevuta in input una lingua dall'utente(italiano, inglese, francese, tedesco o spagnolo), lo saluti nella lingua scelta ("Ciao", "Hello", "Salut", "Tschuss" o "Hola").

```
#include <iostream>
using namespace std;

int main(){
  char lingua;
  cout << "Italiano, inglese, francese, tedesco, spagnolo ? (i|e|f|t|s): ";
  cin >> lingua; lingua = tolower(lingua);
  if (lingua == 'i') cout << "Ciao!"<<endl;
  else if (lingua == 'e') cout << "Hello!" << endl;
  else if (lingua == 'f') cout << "Salut!"<< endl;
  else if (lingua == 't') cout << "Tschuss!"<< endl;
  else if (lingua == 's') cout << "Hola!"<< endl;
  else cout << "Scusa, non parlo la tua lingua." << endl;
  system("PAUSE");
}
```

Tut02.2 Scrivere un programma che, presi in input due punti del piano distinti, restituisca la loro distanza euclidea. (Usando le funzioni)

```
#include <math.h>  // per avere la funzione sqrt()
#include <iostream>
using namespace std;

void distanzaeuclidea(double a, double b, double c, double d){
     if (a==c && b== d) cout << "I punti coincidono \n";
     else{
         double x = sqrt((a-c)*(a-c) + (b-d)*(b-d));
         cout << "Distanza: " <<  x << endl;
         }
     }

int main(){
double x1,x2,y1,y2;
cout << "Inserire i due punti del piano di cui calcolare la distanza:" << endl;
  cout << "x1: ";
  cin >> x1;
  cout << "y1: ";
  cin >> y1;
```

```
  cout << "x2: ";
  cin >> x2;
  cout << "y2: ";
  cin >> y2;

  distanzaeuclidea(x1,y1,x2,y2);


  system("PAUSE");

}
```

Tut02.3 Scrivere un programma che, usando le funzioni, calcoli le soluzioni di un'equazione di secondo grado i cui coefficienti sono dati in input:

a. richiedendo all'utente di inserire parametri per cui l'equazione abbia due soluzioni reali distinte;

```
#include <cmath>  // definisce la funzione sqrt()
#include <iostream>
using namespace std;

void soluzionireali(double a, double b, double c){
       double d = b*b - 4*a*c;
       double sqrtd = sqrt(d);
       double x1 = (-b + sqrtd)/(2*a);
       double x2 = (-b - sqrtd)/(2*a);

       cout << "Le soluzioni sono:" << endl;
       cout << "x1 = " << x1 << endl;
       cout << "x2 = " << x2 << endl;

       return;
     }

int main()
{ // inserisce i coefficiente dell'equazione
  double a, b, c;
  cout << "Inserire i coefficienti dell'equazione tali che a>0 e
(b*b - 4ac)>0:" << endl;
  cout << "a: ";
  cin >> a;
  cout << "b: ";
  cin >> b;
  cout << "c: ";
  cin >> c;
  cout << "Equazione: " << a << "*x*x + " << b << "*x + " << c <<
" = 0" << endl;

  soluzionireali(a,b,c); //calcola le due soluzioni reali
```

```
  system("PAUSE");
}
```

b. considerando anche i casi delle soluzioni coincidenti e immaginarie.

```
#include <cmath>  // definisce la funzione sqrt()
#include <iostream>
using namespace std;

void primogrado(double a, double b){
       if (a == 0 && b != 0) cout << "Impossibile" << endl;
       else if (a == 0 && b == 0) cout << "Indeterminata" << endl;
            else {cout << "Ha una sola soluzione:" << endl;
            cout << "x = " << b / a << endl;
            }
       return;
     }

void soluzionireali(double a, double b, double c){
       double d = b*b - 4*a*c;
       double sqrtd = sqrt(d);
       double x1 = (-b + sqrtd)/(2*a);
       double x2 = (-b - sqrtd)/(2*a);

       cout << "Le soluzioni sono:" << endl;
       cout << "x1 = " << x1 << endl;
       cout << "x2 = " << x2 << endl;

       return;
     }

void soluzionicoincidenti(double a, double b){
       double x = (- b)/(2*a);
       cout << "Le due soluzioni sono coincidenti:" << endl;
       cout << "x1 = x2 = " << x << endl;
       return;
     }

void soluzionicomplesse(double a, double b, double c){
 double d = -b*b + 4*a*c;
 double sqrtd = sqrt(d);

 cout << "Le soluzioni sono:" << endl;
 cout << "x1 = " << -b/(2*a)<<" +" << sqrtd/(2*a) << "i" << endl;
 cout << "x2 = " << -b/(2*a)<<" " <<- sqrtd/(2*a) << "i" << endl;
 return;
     }


int main() { // inseriamo i coefficienti dell'equazione
  double a, b, c;
```

```
  cout << "Inserire i coefficienti dell'equazione:" << endl;
  cout << "a: ";
  cin >> a;
  cout << "b: ";
  cin >> b;
  cout << "c: ";
  cin >> c;
  cout << "Equazione: " << a << "*x*x + " << b << "*x + " << c <<
" = 0" << endl;

  if(a==0) primogrado(b,c); //è un'equazione di primo grado
  else {
       if (b*b - 4*a*c > 0) soluzionireali(a,b,c);
// due soluzioni reali distinte
       else if (b*b - 4*a*c == 0) soluzionicoincidenti(a,b);
// due soluzioni reali coincidenti
       else soluzionicomplesse(a,b,c);
// due soluzioni complesse coniugate
  }
  system("PAUSE");
}
```

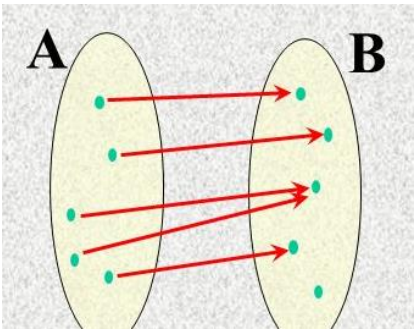
Tut02.4 Scrivete un programma che, inseriti i numeri (a1, b1, c1, a2, b2, c2), coefficienti di due rette a1*x + b1*y + c = 0, a2*x + b2*y + c = 0, dica se (a1, b1, c1, a2, b2, c2) sono i coefficienti di due rette, e se queste rette sono parallele, perpendicolari, incidenti o coincidenti. (Usate le funzioni).

```
#include <iostream>
using namespace std;


void tiporette(double a1, double b1, double c1, double a2, double
b2, double c2){
     double m1, m2, q1, q2;
     if (b1==0 && b2==0)
      {
        if (a1==0 || a2==0) cout << "Non sono due rette" << endl;
        else {if ((c1/a1)!=(c2/a2))
               cout<<"Le rette sono parallele"<< endl;
              else
               cout<<"Le rette sono coincidenti"<< endl;
                    }
               }
     if (b1==0 && b2!=0){
        if (a1==0)
          cout<< "Non sono due rette" <<endl;
         else{if(a2==0)
               cout<<"Le rette sono perpendicolari" << endl;
              else
```

```
                cout<<"Le rette sono incidenti"<< endl;
                    }
                }
     if (b1!=0 && b2==0){
                if (a2==0) cout<< "Non sono due rette" <<endl;
                else{
                     if(a1==0)
                      cout<<"Le rette sono perpendicolari"<<endl;
                     else
                      cout<<"Le rette sono incidenti"<< endl;
                     }
                }
     if (b1!=0 && b2!=0){   //calcolo m e q delle due rette
                m1= -(a1/b1);
                m2= -(a2/b2);
                q1= c1/b1;
                q2= c2/b2;
                if (m1 == m2){
                   if (q1 == q2)
                    cout<<"Le rette sono coincidenti"<< endl;
                   else
                    cout<<"Le rette sono parallele"<<endl;
                   }
                else{
                   if(m1==-(1/m2))
                    cout<<"Le rette sono perpendicolari"<< endl;
                   else cout<<"Le rette sono incidenti"<<endl;
                   }
                }
}


int main(){
    double a1,b1,c1,a2,b2,c2,m1,m2,q1,q2;
//lettura dei coefficienti
    cout<<"Inserisci i coefficienti della prima retta:\n";
    cout<<"a = "; cin>>a1;
    cout<<"b = "; cin>>b1;
    cout<<"c = "; cin>>c1;
    cout<<"Inserisci i coefficienti della seconda retta:\n";
    cout<<"a = "; cin>>a2;
    cout<<"b = "; cin>>b2;
    cout<<"c = "; cin>>c2;
    tiporette(a1, b1, c1, a2, b2, c2);
    system("PAUSE");
}
```

Settimana 03 – Funzioni con valore di ritorno "Funzioni come in matematica"

1.Lezione 05: funzioni con valore di ritorno, traccia esecuzione, composizioni di funzione, overloading (cap.5.1-5.4).
2.Lezione 06: funzioni e espressioni booleane.
3.Tutorato 03: funzioni con valore di ritorno.



In matematica, una funzione prende un oggetto in un insieme A e restituisce un oggetto in un insieme B

Lezione 05 - Funzioni con valore ritorno
Parte 1. Esempi (Cap. 5 libro di testo)

Lezione 05. Prima ora. Nelle lezioni precedenti abbiamo visto funzioni come sqrt(x), che indicano un valore, in questo caso la radice di x. Questo valore viene detto ***valore di ritorno***. Si dice che queste funzioni ***"restituiscono"*** un valore, appunto il loro valore di ritorno. Abbiamo imparato a definire funzioni che ***non*** restituiscono un valore, come **void** NewLine(){cout << endl:}. È possibile definire funzioni che restituiscono un valore, usando nella definizione di funzione il comando

return v;

che ha due effetti: ***termina l'esecuzione*** della funzione e ***restituisce il valore*** v. E' molto importante saper distinguere tra le funzioni con valore di ritorno e quelle senza, perche' i loro risultati compaiono in parti diverse del programma. Piu' esattamente:

1. Una funzione f(x) con valore di ritorno denota un valore, che ***di solito viene utilizzato da altre funzioni*** all'interno di un programma. Questo valore ***non viene automaticamente stampato***: dunque se scriviamo

   f(3);

   il valore di f(3) viene calcolato e poi va perso. Possiamo usare f per construire espressioni, come f(log(5))+1.
2. Una funzione p(x) senza valore di ritorno hanno tipo **void**, non denota un valore, di solito ***stampa qualche cosa sullo schermo***, dunque comunica con noi, oppure modifica la memoria del computer. ***Non può venir stampata***: un comando

   cout << p(x);

   produce un errore. ***Non possiamo usare p per costruire espressioni***: p(3)+1 produce un errore.

Ora vediamo gli esempi del libro di testo.

```
// Lezione05-FunzioniConValoreRitorno
//PRIMA ORA: valori di ritorno, traccia esecuzione,
//composizioni di funzione, overloading (cap.5.1-5.4).
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;
```

```
void stampaarea (double radius)
 {double pi    = acos (-1.0);
  double area = pi * radius * radius;
  cout << area;}
/* il valore viene stampato, non è un valore "restituito".
Stampaarea(x) non denota un valore. */

double area (double radius)
 {double pi    = acos (-1.0);
  double area = pi * radius * radius;
  return area;}
/* area(R) denota l'area del cerchio di raggio R. "return"
interrompe l'esecuzione e "restituisce" il valore dell'area.
Questo valore non viene automaticamente stampato */

/* Un altro esempio: il valore assoluto */
double absoluteValue (double x)
{
 if (x < 0)
  {return -x;} //risultato -x
 else //x>=0
  {return +x;} //risultato +x
}

/* Se la funzione ha un valore di ritorno, dobbiamo ricordarci di
definire un valore in tutti i casi. Ecco un esempio tipico di
definizione errata. */
double absoluteValue2 (double x)
{if (x < 0)
  {return -x;}
 else if (x > 0)
  {return x;}
//ERRORE: se x=0 non viene definito quale valore restituire
}

/* Normalmente non vediamo sullo schermo quali passi fa una
funzione per calcolare un valore. Se tuttavia vogliamo saperlo,
possiamo stampare ogni calcolo intermedio: questa viene chiamata
TRACCIA dell'esecuzione della funzione.
 Questa funzione calcola la distanza tra due punti. */
double distance (double x1, double y1, double x2, double y2)
{double dx = x2 - x1;
 double dy = y2 - y1;
 //TRACCIA esecuzione: da togliere finito di scrivere il programma
```

```
 // cout << "dx = " << dx << " dy = " << dy << endl;
 double dsquared = dx*dx + dy*dy;
 //TRACCIA esecuzione: da togliere finito di scrivere il programma
 // cout << "dsquared = " << dsquared << endl;
 double result = sqrt (dsquared);
 return result;}

/* Una funzione puo' essere definita usando un'altra funzione.
Possiamo calcolare l'area di un cerchio dati il centro e un punto
sulla circonferenza usando la funzione distanza per calcolare il
raggio. */

double cerchio(double xc, double yc, double xp, double yp)
{
 double radius = distance (xc, yc, xp, yp);
 double result = area (radius);
 return result;
}

/* nella prossima versione della funzione "cerchio" riutilizziamo
il nome "area". Questo si puo' fare finche' le due versioni hanno
numero o tipo degli argomenti differente e quindi non possono
essere confuse. Questo riutilizzo si chiama OVERLOADING e serve
per sottolineare che le due funzioni sono simili */
double area(double xc, double yc, double xp, double yp)
{double radius = distance (xc, yc, xp, yp);
 double result = area (radius);
 return result;}
/* Un nome viene riutilizzato per sottolineare la somiglianza tra
funzioni: entrambe le funzioni "area" calcolano un'area. */

int main () /* main di prova delle funzioni definite */
{double x;

 cout << endl << "PROVA stampaarea(x)" << endl;
 x = +3;
 stampaarea(x); /* Stampa l'area del cerchio di raggio 3. Questo è
il modo corretto di usare stampaarea. Di seguito vediamo invece
due errori tipici. */
 // double y = stampaarea(x);
 /* ERRORE 1: stampaarea non denota un valore, non "restituisce"
nulla, quindi non c'è nulla da assegnare a y */
 // cout << stampaarea(x);
 /* ERRORE 2: stampaarea non denota un valore, non "restituisce"
nulla, quindi non c'è nulla da stampare */
```

```
 cout << endl << "PROVA area(x)" << endl;
 x = +3;
 cout << "x = " << x << " area(x) = " << area(x) << endl;

 x = -3;
 cout << "x = " << x << " area(x) = " << area(x) << endl;
 x = 0;
 cout << "x = " << x << " area(x) = " << area(x) << endl;
 area(+1000); /* Anche questo è un ERRORE (anche se non segnalato
come tale): questo valore non è automaticamente stampato, e se non
viene assegnato a una variabile viene perso */

 /* Proviamo la definizione corretta di valore assoluto */
 cout << endl << "PROVA absoluteValue(x)" << endl;
 x = +3;
 cout << "x = " << x << " |x| = " << absoluteValue(x) << endl;
 x = -3;
 cout << "x = " << x << " |x| = " << absoluteValue(x) << endl;
 x = 0;
 cout << "x = " << x << " |x| = " << absoluteValue(x) << endl;

 /* Proviamo la definizione errata di valore assoluto */
 cout << endl << "PROVA absoluteValue2(x)" << endl;
 x = +3;
 cout << "x = " << x << " |x| = " << absoluteValue2(x) << endl;
 x = -3;
 cout << "x = " << x << " |x| = " << absoluteValue2(x) << endl;
 x = 0;
 cout << "x = " << x << " |x| = " << absoluteValue2(x) << endl;
/* Nella definizione errata, |x| non viene definito per x=0. Come
conseguenza, il risultato ottenuto per |0| viene scelto in modo
arbitrario, e puo' essere (ma per caso) proprio 0 */

/* Proviamo altre funzioni definite in modo corretto. */
 cout << endl << "PROVA DISTANCE" << endl;
 cout << "distance(1,1,4,5)=" << distance(1.,1.,4.,5.) << endl;

 cout << endl << "PROVA (AREA) CERCHIO" << endl;
 cout << "cerchio(1,1,4,5)=" << cerchio(1.,1.,4.,5.) << endl;

 cout << endl << "PROVA FUNZIONE OVERLOADED: AREA" << endl;
 cout << "area(1,1,4,5)=" << area(1.,1.,4.,5.) << endl;
 cout << endl; system("pause");}
```

Lezione 05 - Funzioni con valore ritorno
Parte 2. Esercizi

Lezione05. ***Seconda ora: esercizi su funzioni e IF.***

ATTENZIONE. Tutte le funzioni richieste negli esercizi che seguono hanno valori di ritorno, quindi hanno bisogno di una istruzione **return** e non stampano. Per vederne i valori, quindi dovete richiamarle nel **main** usando una istruzione **cout.** Ricordate che **bool** è il tipo dei valori di verità: questo tipo ha due soli elementi, 0=falso e 1=vero.

5.1. Scrivete una funzione

bool divide(**int** d, **int** n)

che prenda in input due interi d, n e restituisca un "booleano": 1=vero se d divide n e 0=falso altrimenti. **Suggerimento.** Usate un IF e l'operazione n%d per calcolare il resto della divisione tra n e d: d divide n se il resto della divisione tra n e d vale 0. **Esempi.** Controllate che divide(2,10)=1 (2 divide 10) e che divide(3,10)=0 (3 non divide 10).

5.2. Scrivete una funzione

int massimo(**int** x, **int** y)

che chieda due interi x, y in input e restituisca il massimo. **Suggerimento.** Usate un IF. **Esempi.** Controllate che massimo(3,2)=3 e che massimo(3,5)=5.

5.3. Scrivere una funzione

bool multipli**(int** a, **int** b)

che chieda in input due interi a, b e restituisca vero se a è multiplo di b oppure viceversa, e falso se a non è multiplo di b e contemporaneamente b non è multiplo di a. **Suggerimento.** riutilizzate la funzione divide(.,.) che controlla se a divide b. Se a divide b rispondete vero, altrimenti controllate se b divide a. **Esempi.** Controllate che multipli(3,2)=0 (2 non divide 3 e 3 non divide 2), e che multipli(4,2)=multipli(2,4)=1.

5.4. Scrivete una funzione

int minimo(**int** x, **int** y, **int** z, **int** t)

che prende in input quattro interi e ne restituisce il minimo. **Suggerimento.** Inserite il minimo in una variabile min utilizzando il test < e 3 IF tronchi. Definite una variabile min=x, e assegnate alla variabile min di volta il volta il piu' piccolo

degli interi che trovate. Il minimo tra x e y è b se (y<min) è vero, altrimenti min resta x e non dovete fare niente. Una volta assegnato a min il minimo tra x, y, confrontate min con z. Se (z<min) il minimo è z, altrimenti il minimo resta min. Infine, se (t<min) il minimo tra x,y,z,t, è t, altrimenti resta min. Notate che non è necessario considerare il caso in cui due o piu' numeri tra x,y,z,t sono uguali: il metodo dato funziona in ogni caso, perche' il ragionamento che ci sta dietro garantisce così. **Esempi.** Definite quattro esempi, uno per ogni possibile posizione del minimo: 0,1,2,3 e 1,0,2,3 e 1,2,0,3 e 1,2,3,0.

Lezione 05 - Funzioni con valore ritorno
Parte 3. Soluzioni

```
// Lezione05-FunzioniConValoreRitorno-Soluzioni
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

/* 5.1 (divisibilita') */
bool divide(int d, int n)
 {
  if (n%d==0)
   {return 1;}
  else /*n%d diverso da 0*/
   {return 0;}
 }

/* 5.2. Massimo di due interi */
int massimo(int x, int y)
{
  if (x < y)
   {return y;}
  else //x>=y
   {return x;}
}

/* 5.3.Controllo se due numeri sono uno il multiplo dell'altro*/
bool multipli(int a, int b)
{
  if (divide(a,b)==1)
   {return 1;}
  else //la risposta è vero se e solo se b divide a
   {return divide(b,a);}
}

/* 5.4 (minimo di quattro numeri) */
int minimo(int x, int y, int z, int t)
{ int min = x;
  if (min > y)
   {min=y ;}
  if (min > z)
   {min=z ;}
  if (min > t)
   {min=t ;}
  return min;}

int main (){int n,d, x,y,z,t;

cout << endl << "PROVA della funzione: divide" << endl;
```

```
n = 3; d=2; cout << " n = " << n << " d = " << d << " divide(d,n)
= " << divide(d,n) << endl;
n = 4; d=2; cout << " n = " << n << " d = " << d << " divide(d,n)
= " << divide(d,n) << endl;

cout << endl << "PROVA della funzione: massimo" << endl;
x = 3; y=2; cout << " x = " << x << " y = " << y << " massimo(x,y)
= " << massimo(x,y) << endl;
x = 3; y=5; cout << " x = " << x << " y = " << y << " massimo(x,y)
= " << massimo(x,y) << endl;

cout << endl << "PROVA della funzione: multipli" << endl;
x = 3; y=2; cout << " x = " << x << " y = " << y << "
multipli(x,y) = " << multipli(x,y) << endl;
x = 4; y=2; cout << " x = " << x << " y = " << y << "
multipli(x,y) = " << multipli(x,y) << endl;
x = 2; y=4; cout << " x = " << x << " y = " << y << "
multipli(x,y) = " << multipli(x,y) << endl;

cout << endl << "PROVA della funzione: minimo" << endl;
x = -1; y=1; z=2; t=3; cout << " x = " << x << " y = " << y << " z
= " << z << " t = " << t << endl;
cout << "minimo(x,y,z,t) = " << minimo(x,y,z,t) << endl;

x = 1; y=-2; z=2; t=3; cout << " x = " << x << " y = " << y << " z
= " << z << " t = " << t << endl;
cout << "minimo(x,y,z,t) = " << minimo(x,y,z,t) << endl;

x = 1; y=2; z=-3; t=3; cout << " x = " << x << " y = " << y << " z
= " << z << " t = " << t << endl;
cout << "minimo(x,y,z,t) = " << minimo(x,y,z,t) << endl;

x = 1; y=2; z=3; t=-4;
cout << " x = " << x << " y = " << y << " z = "
  << z << " t = " << t << endl;
cout << "minimo(x,y,z,t) = " << minimo(x,y,z,t) << endl;
cout << endl; system("pause");}
```

Lezione 06 – Funzioni e espressioni booleane Parte 1. Esempi (Cap. 5.5-5.9 libro testo)

Lezione 06. Prima ora. In questa lezione ci dedicheremo a definire funzioni con valore di ritorno di tipo **bool**, dunque 0 (falso) oppure 1 (vero). In matematica, si tratta semplicemente di funzioni con codominio l'insieme {0,1}. Le funzioni booleane sono molto usate in programmazione, e vengono usate per prendere una scelta tra due possibilità. Utilizzando una istruzione ***if*** *(f(x)==1) {…}* ***else*** *{…}*, quando f(x) vale 1 eseguiamo il primo gruppo di istruzioni, e quando f(x)==0 il secondo.
Introduciamo inoltre le operazioni logiche booleane: congiunzioni, disgiunzioni e negazioni, che ci consentono di definire espressioni booleane complesse a partire da espressioni booleane più semplici.

- Una **negazione** di una espressione booleana A vale 1 (vero) se A vale falso, e vale 0 (falso) se A vale vero.
- Una **congiunzione** di espressioni booleane A, B vale 1 (vero) se sia A che B valgono vero. Vale 0 (falso) se A vale falso, oppure B vale falso, oppure entrambi.
- Una **disgiunzione** di espressioni booleane A, B vale 1 (vero) se A vale vero, oppure B vale vero, oppure entrambi. Vale 0 (falso) se sia A che B valgono falso.

Useremo le operazioni booleane sia per definire condizioni piu' complesse all'interno di un if, che per definire funzioni booleane.

```
// Lezione06-FunzioniBooleane
//PRIMA ORA: il tipo Bool: valori, variabili, operatori, funzioni
// (cap.5.5-5.9 del libro di testo).
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

int n = 7; double x = 3.0;
bool evenFlag     = (n%2 == 0);  // vero se n è pari
bool positiveFlag = (x > 0);     // vero se x è positivo
```

```
//Questa funzione decide se x è una singola cifra: 0, 1, …, 9
bool isSingleDigit1 (int x)
{
 if (x >= 0)
  {
   if (x<10)
    {return true;}
   else
    {return false;}
  }
 else
  {return false;}
}

/* La stessa funzione si puo' definire in modo piu' conciso usando
una congiunzione logica */

bool isSingleDigit2 (int x)
{//la congiunzione logica di A, B si indica con (A && B)
 if ((x >= 0) && (x < 10))
  {return true;}
 else
  {return false;}
}

bool noSingleDigit1 (int x)
{//la negazione logica di A si indica con !A
 return !isSingleDigit1(x);
 //questo comando restituisce il valore di verita'
 //opposto a quello di isSingleDigit1(x)
}

//La funzione noSingleDigit1 si puo' anche definire
//usando una disgiunzione

bool noSingleDigit2 (int x)
{//la disgiunzione logica di A, B si indica con (A || B)
 if ((x < 0) || (x >= 10))
  {return true;}
 else
  {return false;}
}
```

```
int main ()
{cout << " n = " << n << " booleano (n%2 == 0) = "
 << evenFlag << endl;
 cout << " x = " << x << " booleano (x    > 0) = "
 << positiveFlag << endl;

if (evenFlag==1)
 {cout << " n = " << n << " pari" << endl;}
else
  {cout << " n = " << n << " dispari" << endl;}

n=3;
cout << " n = " << n << " isSingleDigit1(x) = "
 << isSingleDigit1(n) << " isSingleDigit2(x) = "
 << isSingleDigit1(n) << endl;

n=33;
cout << " n = " << n << " isSingleDigit1(x) = "
 << isSingleDigit1(n) << " isSingleDigit2(x) = "
 << isSingleDigit1(n) << endl;

if (isSingleDigit2 (n) == 1)
 {cout << " n = " << n << " ha una singola cifra" << endl;}
else
 {cout << " n = " << n << " non ha una singola cifra" << endl;}

cout << "Prova comando system(\"CLS\")" << endl; system("pause");
/* introduciamo ora il comando "CLS" or "Clear Screen" che
cancella completamente lo schermo: ci servira' in un esercizio */
system("CLS");  /*abbiamo bisogno di una seconda pausa per
controllare l'effetto di "CSL" */
cout << "system(\"CLS\") ha cancellato ogni scritta precedente"
<< endl; system("pause");}
```

Lezione 06 - Funzioni e espressioni booleane
Parte 2. Esercizi

Lezione06-FunzioniBooleane. ***SECONDA ORA: esercizi.***

Qualche avvertenza.

- Quando scrivete una espressione logica, per evitare ambiguita', **non saltate parentesi**. Per esempio: scrivete ((x < 0) || (x >= 10)) e non: x<0 || x>=10.
- Il primo e il terzo esercizio (stampaeta e MorraCinese) richiedono funzioni **void**, che stampano dei valori e non usano il **return**.
- Per il secondo esercizio (valoreDiMezzo) è l'opposto: viene richiesta una funzione che da' un valore di ritorno, quindi usa il **return**, e non stampa.
- Gli argomenti di una funzione vengono sempre forniti tramite la "chiamata" della funzione dal main: per esempio con stampaeta(20); Non dovete quindi inserire una richiesta cin >> x; all'interno della definizione di **void** stampaeta(**int** x), perche' facendo inserire dall'utente un valore di x, cancellereste il valore scelto dal main.

6.1. Scrivere una funzione

void stampaeta(**int** x);

che chieda prenda una eta' x in anni e stampi:
- "sei un minorenne" se l'eta' è < 18,
- "sei un adulto" se l'eta è >=18 e < 65",
- "sei un anziano" se l'eta' è > 65. **Suggerimento.** Usate un ***"if … else if … else if …… else …"*** per scegliere tra i 3 casi e una congiunzione logica nel caso di mezzo. **Esempi.** Trovate voi stessi degli esempi.

6.2 Scrivete una funzione

int valoreDiMezzo(**int** x, **int** y, **int** z)

che prende tre numeri interi in input e restituisce in output il loro valore di mezzo. **Suggerimento.** Scrivete una condizione con l'operazione oppure (||) e congiunzioni (&&) che descrive quando x è il valore di mezzo. x è il valore di mezzo se:
- è sia >=y che <=z, oppure se
- è sia <=y che >=z.

Se x è il valore di mezzo restituite x, altrimenti passate a considerare y, altrimenti restituite z. Usate un ***"if … else if …***

else if …… ***else*** …" per scegliere tra i 3 casi. **Esempi.** Inventate tre esempi, uno per ogni possibile posizione del valore di mezzo tra i tre numeri.

6.3 Scrivere una funzione

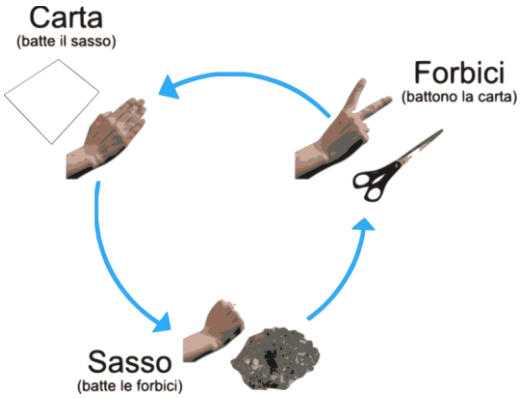
void MorraCinese(){…}

che faccia da giudice di gara per una partita di morra cinese. A due giocatori viene richiesta in input una mossa compresa tra "Sasso" "Carta", "Forbice", senza dire al secondo cosa ha mosso il primo. Il programma scrive in output il nome del vincitore del gioco, determinato in base alle regole seguenti: "carta domina sasso che domina forbice che domina carta". Scelte uguali comportano la parita' del gioco.

Suggerimenti.

- Usate i caratteri **'S'**, **'C'**, **'F'** per rappresentare le scelte, due variabili mossa1, mossa2 di tipo **char** per contenere le mosse dei due giocatori e degli **else if** annidati per calcolare il risultato della partita.
- Usate le istruzione **cin** >> mossa1; e **cin**>> mossa2; che fermano l'esecuzione e aspettano che i valori delle variabili mossa1 e mossa2 vengano introdotte da tastiera.
- Pulite lo schermo tra un **cin** e l'altro, per evitare che il secondo giocatore sappia cosa ha mosso il primo: per farlo, usate **system**("CLS");
- Attenti a non dimenticare le virgolette ' ' in 'S', 'C', 'F' (indicano un carattere) e le doppie virgolette in "CLS" (indicano un testo). Se le dimenticate, S, C, F, CLS vengono prese per variabili del C++, mentre non lo sono.
- Le istruzioni **cin** >> mossa1; e **cin** >> mossa2; devono venire scritti dentro la definizione di MorraCinese(), non nel main: ricordatevi che una funzione non conosce i valori delle variabili dichiarate nel main o in un'altra funzione.

Esempi. Provate a giocare qualche partita.



Lezione 06 - Funzioni booleane
Parte 3. Soluzioni

```
// Lezione06-FunzioniBooleane-Soluzioni

/* 6.1 Maggiorenne e minorenne */
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

void stampaeta(int x)
{
  if (x < 18)
   cout << " sei minorenne " << endl;
  else if (x > 64)
   cout << " sei anziano " << endl;
  else cout << " sei adulto " << endl;
}


/* 6.2 Valore di mezzo tra 3 interi */
int valoreDiMezzo(int x, int y, int z)
{
 if ( ((y <= x) && (x <= z)) || ((z <= x) && (x <= y)) )
   {return x;}
 else if ( ((x <= y) && (y <= z)) || ((z <= y) && (y <= x)) )
   {return y;}
 else
   {return z;}
}


/* 6.3 Morra Cinese */
#include <stdlib.h>
#include <iostream>
#include <math.h>
using namespace std;

void MorraCinese()
{//per leggibilità introduciamo dei nomi per sasso, carta, forbice
 char sasso = 'S';
 char carta = 'C';
 char forbice = 'F';
 char mossa1;
 char mossa2;
cout << "Giocatore1, inserisci sasso (S), carta(C) o forbice(F): "
<<  endl;
cin >> mossa1;
```

```
system("CLS"); //pulisce lo schermo
cout << "Giocatore2, inserisci sasso (S), carta(C) o forbice(F): "
<<  endl;
cin >> mossa2;
system("CLS"); //pulisce lo schermo
cout << " mossa1=" << mossa1 << " mossa2=" << mossa2 << endl;
// PARITÀ
if      ((mossa1 == sasso)   && (mossa2 == sasso))
 cout << " pari " << endl;
else if ((mossa1 == carta)   && (mossa2 == carta))
 cout << " pari " << endl;
else if ((mossa1 == forbice) && (mossa2 == forbice))
 cout << " pari " << endl;
// VINCE GIOCATORE 1
else if ((mossa1 == sasso)   && (mossa2 == forbice))
 cout << " vince Giocatore1 " <<endl;
else if ((mossa1 == carta)   && (mossa2 == sasso))
 cout << " vince Giocatore1 " <<endl;
else if ((mossa1 == forbice) && (mossa2 == carta))
 cout << " vince Giocatore1 " <<endl;
// VINCE GIOCATORE 2
else if ((mossa2 == sasso)   && (mossa1 == forbice))
 cout << " vince Giocatore2 " <<endl;
else if ((mossa2 == carta)   && (mossa1 == sasso))
 cout << " vince Giocatore2 " <<endl;
else if ((mossa2 == forbice) && (mossa1 == carta))
 cout << " vince Giocatore2 " <<endl;
// MOSSE NON REGOLARI
else
 cout << " mosse non regolari " <<endl;
}
// SONO POSSIBILI MOLTI ALTRI MODI DI STABILIRE IL VINCITORE
// E ALCUNI SONO DECISAMENTE PIÙ BREVI

//Proviamo le funzioni definite in precedenza
int main()
{cout << "valoreDiMezzo(1,2,3)=" << valoreDiMezzo(1,2,3)
  << endl;
 cout << "valoreDiMezzo(1,3,2)=" << valoreDiMezzo(1,3,2)
  << endl;
 cout << "valoreDiMezzo(2,1,3)=" << valoreDiMezzo(2,1,3)
  << endl;
 system("pause");

 system("CLS"); //pulisce lo schermo

 MorraCinese();
 system("pause");}
```

Tutorato 03
Parte 1. Esercizi proposti

Come sempre, ricordatevi ***concentrarvi sui primi esercizi***.

Tut03.1 Scrivere un programma *(tutto nel main, senza usare funzioni)* che riceva in input quattro numeri interi e calcoli la somma dei numeri pari e quella dei numeri dispari.

Tut03.2 Scrivere un programma *(tutto nel main, senza usare funzioni)* che riceva in input tre numeri interi e li scriva in output in modo ordinato.

Tut03.3 Scrivere un programma che calcoli la retta per due punti dati in input (ce n'è sempre almeno una), e stampi il risultato. ***Vi chiediamo di risolvere il problema con una funzione void.***

Tut03.4 Scrivere un programma che calcoli la distanza di due punti del piano dati in input. ***Vi chiediamo di definire una funzione che restituisca la distanza.***

Tut03.5 Data un'operazione con due operandi interi (ad esempio *3 + 4*, con uno spazio tra operandi ed operatore), calcolarne e visualizzarne il risultato (in questo caso, *3 + 4 = 7*). Le operazioni supportate sono: somma (*+*), sottrazione (*-*), prodotto (***), divisione (*/*) e modulo (*%*). Per ognuna di queste operazioni, ***vi chiediamo di scrivere una funzione con valore di ritorno che la calcoli***: ogni funzione dovrà prendere in argomento i due operandi, e restituire il risultato. Invece la richiesta degli argomenti all'utente deve essere fatta dal main: ***abituatevi a svolgere input/ouput e calcolo separatamente***. Nel caso di divisione o modulo per zero (ossia nel caso in cui il secondo operando di queste operazioni sia *0*), non procedere con l'operazione, ma stampare a video un messaggio d'errore. **Suggerimento:** per leggere l'operazione in input, usare l'istruzione **cin** >> m >> op >> n, con m ed n i due numeri, ed op il carattere dell'operazione.

Tutorato 03 – fuzioni con valore di ritorno
Parte 2. Soluzioni

Tut03.1 Scrivere un programma *(tutto nel main, senza usare funzioni)* che riceva in input quattro numeri interi e calcoli la somma dei numeri pari e quella dei numeri dispari.

```
#include <iostream>
using namespace std;
int main() {

     int
a,b,c,d;
     int sommaP, sommaD;
     sommaP = 0;
     sommaD = 0;
     cout << "Inserisci i quattro numeri: ";
     cin >> a >> b >> c >> d;
     if(a%2==0){
          sommaP += a;
     }
     else {
          sommaD += a;
     }
     if(b%2==0){
          sommaP += b;
     }
     else {
               sommaD += b;
     }
     if(c%2==0){
          sommaP += c;
     }
     else {
          sommaD += c;
     }
     if(d%2==0){
          sommaP += d;}
     else {
          sommaD += d;
     }
     cout << "La somma dei numeri pari e " << sommaP << endl;
      cout << "La somma dei numeri dispari e " << sommaD << endl;
       system("PAUSE");
}
```

Tut03.2 Scrivere un programma *(tutto nel main, senza usare funzioni)* che riceva in input tre numeri interi e li scriva in output in modo ordinato.

```
#include <iostream>
```

```
using namespace std
int main() {
int a,b,c;
cout << "Inserisci i tre numeri da ordinare: ";
 cin >> a >> b >> c;

if(a > b){
  if(b > c){

       cout << "Numeri ordinati: " << c << b << a << endl;
  }
  else {

       if (a > c){
            cout << "Numeri ordinati: " << b << c << a << endl;
       }
       else{
            cout << "Numeri ordinati: " << b << a << c << endl;
       }
  }
}
else{
  if(a > c){
       cout << "Numeri ordinati: " << c << a << b << endl;}
  else {
if(b > c){
            cout << "Numeri ordinati: " << a << c << b << endl;
       }
       else{
            cout << "Numeri ordinati: " << a << b << c << endl;
       }
  }
}
system("PAUSE");
}
```

Tut03.3 Scrivere un programma che calcoli una retta per due punti dati in input (ce n'è sempre almeno una)), e stampi il risultato. ***Vi chiediamo di risolvere il problema con una funzione void.***

```
#include <iostream>
using namespace std;

void retta (double x1, double x2, double y1, double y2){
 double m,q;
 if(x1 == x2) cout <<"\nL'equazione della retta è x ="<< x1;
 else{
    m=(y2-y1)/(x2-x1);
    q=-m*x1 + y1;
    cout<<"\nL'equazione della retta è y = "<< m <<"x ";
    if (q>0){
         cout<<"+"<<q << "\n";
```

```
      }
      else{
           cout<< "\n";
      }
  }


}

int main(){
double x1,x2,y1,y2;

  cout<<"Inserire l'ascissa del primo punto: ";
  cin>>x1;
  cout<<"Inserire l'ordinata del primo punto: ";
  cin>>y1;
  cout<<"Inserire l'ascissa del secondo punto: ";
  cin>>x2;
  cout<<"Inserire l'ordinata del secondo punto: ";
  cin>>y2;

  if(x1==x2 && y1==y2) cout << "I due punti coincidono"<<endl;
  else  retta(x1,x2,y1,y2);
  system("PAUSE");

}
```

Tut03.4 Scrivere un programma che calcoli la distanza di due punti del piano dati in input. ***È richiesto di usare una funzione che restituisca la distanza.***

```
#include <math.h>  // è necessario per avere la funzione sqrt()
#include <iostream>
using namespace std;

double distanzaeuclidea(double a, double b, double c, double d){
         return  sqrt((a-c)*(a-c) + (b-d)*(b-d));
      }

int main(){
double x1,x2,y1,y2;
cout << "Inserire i due punti del piano di cui calcolare la
distanza:" << endl;
  cout << "x1: ";
  cin >> x1;
  cout << "y1: ";
  cin >> y1;
  cout << "x2: ";
  cin >> x2;
  cout << "y2: ";
  cin >> y2;
```

```
  cout << "Distanza: " <<  distanzaeuclidea(x1,y1,x2,y2) << endl;


  system("PAUSE");

}
```

Tut03.5 Data un'operazione con due operandi interi (ad esempio *3 + 4*, con uno spazio tra operandi ed operatore), calcolarne e visualizzarne il risultato (in questo caso, *3 + 4 = 7*). Le operazioni supportate sono: somma (*+*), sottrazione (*-*), prodotto (***), divisione (*/*) e modulo (*%*). Per ognuna di queste operazioni, ***vi chiediamo di scrivere una funzione con valore di ritorno che la calcoli***: ogni funzione dovrà prendere in argomento i due operandi, e restituire il risultato. Invece la richiesta degli argomenti all'utente deve essere fatta dal main: ***abituatevi a svolgere input/ouput e calcolo separatamente***. Nel caso di divisione o modulo per zero (ossia nel caso in cui il secondo operando di queste operazioni sia *0*), non procedere con l'operazione, ma stampare a video un messaggio d'errore. **Suggerimento:** per leggere l'operazione in input, usare l'istruzione **cin** >> m >> op >> n, con m ed n i due numeri, ed op il carattere dell'operazione.

```
#include <math.h>
#include <iostream>
using namespace std;
/* Calcola il risultato di un'operazione data, nella forma num op
num. Le operazioni supportate sono: +, -, *, /, %.
*/

//somma di due valori
int somma (int a, int b)
{
return a + b;
}
//differenza di due valori
int diff (int a, int b)
{
return a - b;
}
//prodotto di due valori
int prod (int a, int b)
{
return a * b;
}
//divisione di due valori
double divi (int a, int b)
{
return (double)a / (double)b;
}
//modulo di due valori
```

```
int mod (int a, int b){return a % b;}
int main(){
// il main chiede i valori e le funzioni calcolano il risultato:
// un buon programma che tiene divisi i due aspetti è più facile
// da leggere e da correggere

int m,n;
char op;
cout << "Calcola il risultato di un'operazione nella forma <num>
<op> <num>" << endl;
cout << "Inserisci l'operazione: ";
cin >> m >> op >> n;
if(op == '+')
 cout << m << " " << op << " " << n << " = " << somma(m,n) <<
endl;
else if(op == '-')
 cout << m << " " << op << " " << n << " = "<< diff(m,n) << endl;
else if(op == '*')
 cout << m << " " << op << " " << n << " = " << prod(m,n) << endl;
else if(op == '/' && n != 0)
 cout << " " << op << " " << n << " = " << divi(m,n) << endl;
else if(op == '%' && n != 0)
 cout << m << " " << op << " " << n << " = " << mod(m,n) << endl;
else
 cout << "Operazione non supportata"<< endl;
system("PAUSE");
}
```

Settimana 04 – Ricorsione *(cenni)*
"Come definire un oggetto tramite se stesso"

1. **Lezione 07: cenni di ricorsione**
2. **Lezione 08: ripasso**
3. **Tutorato 04: cenni di ricorsione.**

Il quadro rappresenta una scena che contiene il quadro stesso

Lezione 07 - Cenni di ricorsione
Parte 1.Esempi(Cap.4.7-4.9,5.10-5.12 testo)

Lezione 07. Prima ora. In questa lezione svolgeremo alcuni cenni relativi all'argomento "ricorsione", che si trovano alla fine dei capitoli 4 e 5 del libro di testo. Sappiamo che in matematica una funzione si puo' definire tramite se stessa per ***induzione sul valore***, per esempio quando definiamo il fattoriale con 0! = 1 e (x+1)! = x * x!. Possiamo ricopiare queste definizioni dentro molti linguaggi di programmazione, incluso il C++. La definizione induttiva di una funzione in C++ viene detta ***definizione ricorsiva***, e le funzioni così definite sono dette ***funzioni ricorsive***.

Quando scriviamo una definizione ricorsiva non stiamo spiegando al compilatore C++ ***come calcolare un valore***, ma ci limitiamo a ***definire questo valore*** per induzione. Spetta poi al compilatore generare nel file .exe un programma che calcoli il valore che noi abbiamo definito. In altre parole, la ricorsione è una ***programmazione di secondo livello***, affidata al computer. Noi definiamo il risultato, in questo caso, con una definizione induttiva, e il compilatore trova un programma per calcolarlo. La difficoltà di un programma ricorsivo sta proprio nel fatto che dobbiamo ricordarci di ***non*** inserire istruzioni per calcolare direttamente il risultato, come invece abbiamo fatto finora.

Stampando passo passo i calcoli svolti da un programma ricorsivo avremo un'idea di come un compilatore traduca in programma una definizione induttiva. ***Una descrizione precisa*** di come questo accade, tuttavia, ***non fa parte del corso***.

```
// Lezione07-Cenni di ricorsione
//PRIMA ORA: capitoli 4.7-4.9 e 5.10-5.12 del libro di testo
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;
```

```
/* Definiamo una funzione "countdown ("conto alla rovescia"), che
dato n stampi, in quest'ordine, n, n-1, n-2, …, 2, 1 e poi
"Blastoff" ("partenza!" di un missile). La definizione per
induzione della funzione è: se n=0 stampo "partenza", altrimenti
```

```
stampo n e ricomincio il "conto alla rovescia" da n-1. Ricopiamo
questa definizione in C++. Il compilatore C++ la accetta e la
trasforma in un programma (non sappiamo quale). */


void countdown (int n) //supponiamo n>=0
{
 if (n == 0)
  {cout << "Blastoff!" << endl;}
 else //n>0
  {cout << n << endl;
   countdown (n-1);}
}

/* Definiamo per induzione una funzione che stampi n righe vuote.
La definizione per induzione è: se n=0 faccio nulla, se n>0 stampo
una riga vuota e ricomincio la stampa da n-1. Di nuovo, possiamo
ricopiare  questa definizione in C++. Il compilatore C++ la
accetta e la trasforma in un programma (non sappiamo quale). */

void nLines (int n) //supponiamo n>=0
{if (n > 0)
  {cout << endl;
   nLines (n-1);}/*se (n>0) è falso allora non facciamo nulla*/ }

/* FATTORIALE ottenuto ricopiando la definizione per induzione */
int fact (int n) //supponiamo n>=0
{if (n == 0)
  {return 1;}                  //calcolo !0 = 1
 else //n>0
  {int recurse = fact (n-1); //calcolo !(n-1)
   int result = n * recurse; //calcolo !n = n * !(n-1)
   return result;}}

/* Se chiediamo a una funzione ricorsiva di avvisarci ogni volta
che inizia o finisce il proprio compito, riusciamo a seguire nelle
linee generali il ragionamento fatto dal programma per calcolare
il risultato. Vediamo il caso del fattoriale. */

/* FATTORIALE che stampa passo passo i calcoli svolti */
int factorial (int n) //supponiamo n>=0
{//messaggio con INIZIO dell'esecuzione di factorial(n)
 cout << "Inizio calcolo factorial(" << n << ")" << endl;
 if (n == 0)
  {//messaggio con FINE dell'esecuzione di factorial(0)
```

```
   cout << "factorial(0)=1" << endl;
   return 1;}
 else //n>0
  {int recurse = factorial (n-1);
   int result = n * recurse;
   //messaggio con FINE dell'esecuzione di factorial(n)
   cout << "factorial(" << n  << ")=" << result << endl;
   return result;}}

 /* Facendo stampare passo passo i calcoli svolti dall'esecuzione
di factorial(3), vediamo che il programma generato dal compilatore
esamina la definizione induttiva di factorial(3) come farebbe un
essere umano.
- Si chiede come calcolare factorial(3).
- Per risolvere questo problema cerca di calcolare factorial(2),
- Per risolvere questo problema cerca di calcolare factorial(1)
- Per risolvere questo problema cerca di calcolare factorial(0)
- Sa che factorial(0)=1 per definizione.
- Ne deduce che factorial(1)=1
- Ne deduce che factorial(2)=2
- Ne deduce che factorial(3)=6.
A questo punto il programma termina e restituisce 6.
Nelle prossime lezioni vedremo altri esempi. */

int fibonacci (int n) //supponiamo n>=0
{if (n == 0 || n == 1)
  {return 1;}
 else //n>=2
  {int result = fibonacci (n-1) + fibonacci (n-2);
   return result;}
}

int main ()
{cout << "TEST FUNZIONE: countdown(10)" << endl;
 countdown(10);
 system("pause");
 system("CLS"); //cancella ogni carattere sullo schermo

 cout << "TEST FUNZIONE nLines(10)" << endl;
 nLines(10);
 cout << "Sono comparse 10 righe vuote" << endl;
 system("pause");
 system("CLS"); //cancella ogni carattere sullo schermo

 cout << "TEST FUNZIONE fact(3)" << endl;
```

```
cout << "VALORE fact(3) = " << fact(3) << endl;
system("pause");
system("CLS"); //cancella ogni carattere sullo schermo

cout << "TEST FUNZIONE factorial(3)" << endl;
cout << "VALORE factorial(3) = " << factorial(3) << endl;
system("pause");
system("CLS"); //cancella ogni carattere sullo schermo

cout << "TEST FUNZIONE fibonacci(5)" << endl;
cout << "VALORE fibonacci(5) = " << fibonacci(5) << endl;
system("pause");
system("CLS"); /*cancella ogni carattere sullo schermo*/}
```

Lezione 07 - Cenni di ricorsione
Parte 2. Esercizi

Avvertenze.

- Tutti gli esercizi seguenti richiedono la ricorsione.
- Per risolvere questi esercizi dovete prima ricopiare nel file .cpp che usate le definizioni delle funzioni ***fact*** e ***fibonacci*** viste a lezione, perche' le soluzioni degli esercizi le utilizzano. Non usate la versione ***factorial*** del fattoriale, per evitare un eccesso di messaggi.
- Inoltre, i primi due esercizi richiedono funzioni **void**, che stampano dei valori e non usano il **return**. Per il terzo esercizio è l'opposto: viene richiesta una funzione che da' un valore di ritorno, quindi usa il **return**, e non stampa.
- Inziate ***pensando una definizione induttiva fatto solo di parole***, poi trascrivetela in C++. Alla fine confrontate la vostra definizione induttiva con quella che trovate nelle soluzioni.
- Quando ricopiate in C++ la definizione induttiva di una funzione con valore di ritorno, come il fattoriale, ricordatevi di aggiungere il comando return davanti a ogni caso della definizione induttiva. La definizione matematica non lo richiede, il C++ si'.

7.1. Scrivete una funzione ricorsiva

```
void stampaFatt(int n) {…};
```

che stampi all'indietro i valori della funzione fattoriale da n a 0 compresi. **Suggerimento.** Riscrivete la funzione countdown(n), in modo che al posto dei valori di n stampi i valori di fact(n). Ricordatevi che è un errore scrivere: ~~`cout << stampaFatt(0);`~~

Esempi. Controllate che stampaFatt(0) stampa 1, e che stampaFatt(10) stampa: 3628800, 362880, 40320, 5040, 720, 120, 24, 6, 2, 1, 1.

7.2. scrivete una funzione ricorsiva

```
void stampaFibonacci(int a, int b) {…};
```

che stampi i valori di fibonacci dall'intero a all'intero b compresi, in quest'ordine (e dunque in avanti). **Suggerimento.** Usate la seguente definizione induttiva delle azioni da svolgere: se a>b non stampate nulla, altrimenti stampate fibonacci(a) e chiedete di stampare i valori di fibonacci da a+1 fino a b.
Esempi. Controllate che stampaFibonacci(0,0) stampa 1, e che

stampaFibonacci(0,15) stampa: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987.

7.3. scrivete una funzione ricorsiva con valore di ritorno:

```
int sommaInteri(int n) {…};
```

che sommi gli interi da 0 a n e restituisca il risultato come valore di ritorno. **Suggerimento.** Riutilizzate la definizione del fattoriale, cambiandola in modo tale che anziche' calcolare 1*2*3*…*n calcoli 1+2+3+…+n. **Esempi.** Controllate che sia: sommaInteri(0) = 0 e sommaInteri(10) = 55.

Lezione 07 - Cenni di ricorsione
Parte 3. Soluzioni

Avvertenza. Quando ricopiate la definizione induttiva di una funzione con valore di ritorno, come il fattoriale, ricordatevi di aggiungere il comando return davanti a ogni caso della definizione induttiva. La definizione matematica non lo richiede, il C++ si'.

Definiamo induttivamente la funzione di stampa del fattoriale come segue:

1. **Stampa di n!, …, 0!.** Stampa(n) = stampiamo 0! se n=0, altrimenti stampiamo n! e poi eseguiamo Stampa(n-1) (cioe' stampiamo (n-1)!, …, 0!).
2. **Stampa di fibonacci(a), …, fibonacci(b).** Stampa(a,b) = nessuna azione se a>b, altrimenti stampiamo fibonacci(a) e poi eseguiamo Stampa(a+1,b) (cioe' stampiamo fibonacci(a+1), …, fibonacci(b)).
3. **Somma primi n interi.** Somma(n) = 0 se n=0, altrimenti Somma(n) = Somma(n-1) + n.

Ora traduciamo le definizioni induttive qui sopra in C++.

```
// Lezione07-Cenni di ricorsione
//SECONDA ORA: soluzioni
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

int factorial (int n) //senza TRACCIA dell'esecuzione. Sia n>=0
{if (n == 0)
  {return 1;}
 else
  {int recurse = factorial (n-1);
   int result = n * recurse;
   return result;}
}

int fibonacci (int n)
{if (n == 0 || n == 1)
  {return 1;}
 else
  {int result = fibonacci (n-1) + fibonacci (n-2);
```

```
   return result;}
}

void stampaFatt(int n) //Sol. Es. 7.1. Supponiamo n>=0
{
 if (n == 0)
  {cout << "fattoriale(0)=" << factorial(0) << endl;}
 else //n>0
  {cout << "fattoriale(" << n << ")=" << factorial(n) << endl;;
   stampaFatt(n-1);}
}

void stampaFibonacci(int a, int b) //Sol. Es. 7.2. Supponiamo 0<=a
{
 if (a > b)
  {} //se a>b non ci sono valori da stampare: non faccio nulla
 else //a<=b
  {cout << "fibonacci(" << a << ")=" << fibonacci(a) << endl;;
   stampaFibonacci(a+1,b);}
}

int sommaInteri(int n) //Sol. Es. 7.3.
{if (n == 0)
  {return 0;}
 else
  {int recurse = sommaInteri (n-1);
   int result = n + recurse;
   return  result;} /* se volete un valore di ritorno, quando
ricopiate una definizione induttiva in C++ aggiungete un return;*/
}

int main ()
{
 cout << "TEST stampaFatt(10)" << endl;
 stampaFatt(10);
 system("pause"); system("CLS");

 cout << "TEST stampaFibonacci(0,15)" << endl;
 stampaFibonacci(0,15);
 system("pause"); system("CLS");

 cout << "TEST sommaInteri(10)" << endl;
 cout << " sommaInteri(10)=" << sommaInteri(10) << endl;
 system("pause"); system("CLS");}
```

Lezione 08 – Ripasso e esercizi
Parte 1. Ripasso funzioni

Lezione 08. Prima ora. Nella prima ora rivediamo alcuni concetti di base introdotti nelle lezioni precedenti: funzioni, valori di ritorno, definizioni induttive. Rivediamo anche alcuni errori tipici. Nella prossima ora svolgeremo degli esercizi per ripassare tutto quanto visto finora.

```
// Lezione08-Ripasso-Esercizi su booleani e ricorsione
//PRIMA MEZZ'ORA: ripasso
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

/* Modi corretti e non corretti di definire una funzione VOID
(senza valore di ritorno) */
void stampaquadrato(int x)
{cout << x*x;
 //return x*x;//NO! questa funzione non ha valore di ritorno
 }

/* modi corretti e non corretti di definire una funzione con
valore di ritorno */
int quadrato(int x)
{
// cout << x*x; //ERRORE!
//quadrato deve usare un "return" per
//definire un valore di tipo int
 return x*x;
}

/* Definizione induttiva non corretta: f(x) = 1+f(x-1) manca del
caso base e non definisce nulla. Piu' esattamente: cercare di
darci un senso produce un circolo vizioso, non si arriva mai a
capire quando vale f(x). è quello che capita al compilatore quando
cerca di interpretarla. */
int f(int x)
{return 1+f(x-1);}
```

```
int main ()
{ //Eseguire la definizione induttiva errata f produce un errore
  // cout << "f(0) = " << f(0) << endl;
  //Uso corretto di una funzione VOID
 stampaquadrato(10);

 //Uso NON CORRETTO di funzione VOID
 //1. non si puo' definire una funzione nel main o in un'altra
funzione
 // void NewLine(){cout << endl;}; //<= NO! Siamo dentro il main
 //2.Quando si chiama una funzione non si ripetono i tipi:
 //void stampaquadrato(int 10); //<= NO! Non si ripetono i tipi

 //Uso corretto di funzione con valore di ritorno
 cout << endl << quadrato(quadrato(7)+1) << endl;
 //uso corretto di funzione con valore di ritorno
 int y = quadrato(11);
 cout << y << endl;
 cout << endl;

 //Uso NON CORRETTO di una funzione con valore di ritorno
 //quadrato(20);//non salvato in alcun modo, sparisce
 //Uno dei modi corretti di richiamare quadrato(20)
 cout << quadrato(20) << endl;

//cout << stampaquadrato(20);
//ERRORE:stampaquadrato(20) non ha un valore

 system("PAUSE");}
```

Lezione 08 – Ripasso e Esercizi
Parte 2. Esercizi su booleani e ricorsione

Avvertenze. Ecco alcuni errori comuni che dovreste evitare.

- Il primo esercizio richiede una funzioni **void**, che stampa dei valori e non usa il **return**.
- Per gli altri esercizi è l'opposto: viene richiesta una funzione che da' un valore di ritorno, quindi usa il **return**, e non stampa.
- Gli esercizi 1,6 non richiedono la ricorsione, gli esercizi 2,3,4,5 richiedono la ricorsione.
- Fate attenzione a scrivere ***(b==0) e non (b=0)*** nelle condizioni degli ***if***. In C++ non potete omettere il prodotto: scrivete quindi ***2x e non 2*x***.
- Se definite una funzione con un valore di ritorno, quando ricopiate la definizione induttiva in C++ aggiungete un return;
- Concentratevi sui primi esercizi.

8.1. Scrivere, usando un IF, una funzione

```
void PariDispari(){…};
```

che faccia da giudice di gara per una partita di Pari e Dispari. La funzione non richiede ricorsione. A due giocatori viene richiesta in input un numero intero, senza dire al secondo cosa ha mosso il primo. Il programma scrive in output il nome del vincitore del gioco, determinato in base alle regole seguenti: se la somma dei due interi è pari vince il primo giocatore, altrimenti il secondo. **Suggerimento.** Usate due variabili mossa1 e mossa2 di tipo intero per rappresentare le mosse. Usate le istruzione **cin** >> mossa1; e **cin** >> mossa2; che fermano l'esecuzione e aspettano che i valori delle variabili mossa1 e mossa2 vengano introdotte da tastiera. Pulite lo schermo tra un **cin** e l'altro, per evitare che il secondo giocatore sappia cosa ha mosso il primo: per farlo, usate **system**("CLS");

Attenzione. Le istruzioni **cin** >> mossa1; e **cin** >> mossa2; devono venire scritti dentro la definizione di PariDispari(), non nel main, perché PariDispari() non conosce i valori delle variabili di un'altra funzione o del main.

Esempi. Provate a giocare qualche partita.

8.2. Scrivete una funzione ricorsiva

```
double potenza(double a, int b) {…};
```

che dati un reale a e un intero b>=0 restituisca il reale a^b. Non utilizzate la funzione pow già disponibile in libreria, ma traducete in C++ la seguente definizione induttiva: $a^0=1$, $a^{b+1}=(a^b)*a$. Controllate che sia: $10^0=1$, $10^1=10$, $10^2=100$, $10^3=1000$.

8.3. Scrivete una funzione ricorsiva

```
int SD (int n) {…};
```

che prenda un intero n>=0 e restituisca come valore di ritorno l'intero (1+3+5+…+(2n-1)) (ovvero la somma dei primi n dispari). **Suggerimento.** Traducete in C++ la seguente definizione induttiva: se n=0, allora SD(n)=0, se n>0, allora SD(n)=SD(n-1)+(2n-1). **Esempi.** Controllate che sia: SD(0)=0, SD(1)=1, SD(2)=4, SD(3)=9, SD(4)=16. Ricordatevi che in C++ bisogna scrivere 2*n e non 2n.

8.4. Scrivete una funzione ricorsiva

```
int log(int a, int b) {…};
```

che dati due interi a>=2, b>=1 restituisce $\log_a(b)$ arrotondato per difetto a un valore intero. Non utilizzate la funzione log già disponibile in libreria. **Suggerimento.** traducete in C++ la seguente definizione induttiva del logaritmo intero: se b/a (arrotondato per difetto) vale 0, allora log(a,b)=0, altrimenti log(a,b) = 1 + log(a,b/a). **Esempi.** Controllate che sia: log(3,1)=0, log(3,10)=2, log(3,100)=4, log(3,1000)=6.

8.5. ***(Massimo Comun Divisore).*** Scrivete una funzione ricorsiva **int** MCD(**int** a, **int** b) che calcoli il Massimo Comun Divisore di due interi a≥0, b>0. **Suggerimento.** Traducete in C++ la seguente definizione induttiva di MCD: MCD(a,b) = b se a=0, e MCD(a,b) = MCD(b%a, a) se a>0. **Esempi.** Controllate che sia: MCD(0,11) = 11, MCD(13,18) = 1, MCD(144,111) = 3, MCD(144,96) = 48.

8.6. ***(Minimo comune multiplo).*** Scrivete una funzione mcm(**int** a, **int** b) che calcoli il minimo comune multiplo di due interi positivi a, b. La funzione non richiede ricorsione. **Suggerimento.** Usate la funzione MCD definita in precedenza e la formula mcm(a,b) = a*b/MCD(a,b). **Esempi.** Controllate che sia: mcm(1,11) = 11, mcm(12,5) = 60, mcm(144,48) = 144, mcm(10,15)=30.

Lezione 08 – Ripasso e Esercizi
Parte 3. Soluzioni

```
// Lezione08-Ripasso. Soluzione degli esercizi proposti
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

/* 8.1. Funzione che fa da giudice di gara in una
partita a pari e dispari */
void PariDispari()
{int mossa1, mossa2;
 cout << " Mossa Giocatore1 (un intero) = " << endl;
 cin >> mossa1; system("CLS");
 cout << " Mossa Giocatore2 (un intero) = " << endl;
 cin >> mossa2; system("CLS");
 cout << " Mossa Giocatore1 = " << mossa1 <<
 " Mossa Giocatore2 = " << mossa2 << endl;
 if ((mossa1 + mossa2)%2 == 0)
  {cout << "Somma pari:    vince Giocatore1" << endl;}
 else
  {cout << "Somma dispari: Vince Giocatore2" << endl;}
};

/* 8.2. potenza */
double potenza(double a, int b) //Sia b>=0
{if (b==0)
  {return 1;}
 else //b>0
  {return potenza(a,b-1)*a;} /* se volete un valore di ritorno,
quando ricopiate la definizione induttiva in C++ aggiungete un
return; */
};

/* 8.3. Somma dei primi n dispari.*/
//Attenti a scrivere 2*n-1 e non 2n-1
int SD(int n) //Sia n>=0
{if (n==0)
  {return 0;}
 else //n>0
  {return SD(n-1)+(2*n-1);/*non scrivete 2n+1*/}/* se volete un
valore di ritorno, quando ricopiate la definizione induttiva in
C++ aggiungete un return; */
};
```

```
/* 8.4. parte intera di log(a,b) */
int log(int a, int b) //Siano a>=2, b>=1
{if ((b/a)==0)
  {return 0;}
 else //(b/a)>=1
  {return 1+log(a,b/a);} /* se volete un valore di ritorno, quando
ricopiate la definizione induttiva in C++ aggiungete un return; */
};

/* 8.5. MCD(a,b) */
int MCD(int a, int b) //Siano a>0, b>=0
{if (a==0)
  {return b;}
 else //a>0
  {return MCD(b%a,a);} /* se volete un valore di ritorno, quando
ricopiate la definizione induttiva in C++ aggiungete un return; */
};

/* 8.6. mcm(a,b) */
int mcm(int a, int b) //Siano a>0, b>0
{return (a*b)/MCD(a,b);};


int main (){double x;int a,b,n;
 cout << "TEST PariDispari()" << endl;
 PariDispari();system("PAUSE");system("CLS");

 cout << "TEST potenza(x,n)" << endl;
 x=10;n=0; cout << "potenza(" << x << "," << n << ")="
 << potenza(x,n) << endl;
 x=10;n=1; cout << "potenza(" << x << "," << n << ")="
 << potenza(x,n) << endl;
 x=10;n=2; cout << "potenza(" << x << "," << n << ")="
 << potenza(x,n) << endl;
 x=10;n=3; cout << "potenza(" << x << "," << n << ")="
 << potenza(x,n) << endl;
 system("PAUSE");system("CLS");

 cout << "TEST SD(n)" << endl;
 n=0; cout << "SD(" << n << ")=" << SD(n) << endl;
 n=1; cout << "SD(" << n << ")=" << SD(n) << endl;
 n=2; cout << "SD(" << n << ")=" << SD(n) << endl;
 n=3; cout << "SD(" << n << ")=" << SD(n) << endl;
 n=4; cout << "SD(" << n << ")=" << SD(n) << endl;
```

```
 system("PAUSE");system("CLS");

 cout << "TEST log(a,b)" << endl;
 a=3;b=1; cout << "log(" << a << "," << b << ")="
 << log(a,b) << endl;
 a=3;b=10; cout << "log(" << a << "," << b << ")="
 << log(a,b) << endl;
 a=3;b=100; cout << "log(" << a << "," << b << ")="
 << log(a,b) << endl;
 a=3;b=1000; cout << "log(" << a << "," << b << ")="
 << log(a,b) << endl;
 system("PAUSE");system("CLS");

 cout << "TEST MCD(a,b)" << endl;
 a=0;b=11;cout << "MCD(" << a << "," << b << ")="
 << MCD(a,b) << endl;
 a=13;b=18;cout << "MCD(" << a << "," << b << ")="
 << MCD(a,b) << endl;
 a=144;b=111;cout << "MCD(" << a << "," << b << ")="
 << MCD(a,b) << endl;
 a=144;b=96;cout << "MCD(" << a << "," << b << ")="
 << MCD(a,b) << endl;
 system("PAUSE");system("CLS");

 cout << "TEST mcm(a,b)" << endl;
 a=1;b=11;cout << "mcm(" << a << "," << b << ")="
 << mcm(a,b) << endl;
 a=12;b=5;cout << "mcm(" << a << "," << b << ")="
 << mcm(a,b) << endl;
 a=144;b=48;cout << "mcm(" << a << "," << b << ")="
 << mcm(a,b) << endl;
 a=10;b=15;cout << "mcm(" << a << "," << b << ")="
 << mcm(a,b) << endl;
 system("PAUSE");system("CLS");
}
```

Tutorato 04 – cenni di ricorsione
Parte 1. Esercizi

Per tutti gli esercizi di oggi vi chiediamo di scrivere una soluzione ricorsiva. Prendendo la definizione induttiva della soluzione data come suggerimento e ricopiandola in C++.

Tut04.1 Scrivete una funzione ricorsiva

```
void stampalinea(int n){…};
```

che preso in input un numero naturale n>=0 stampi una linea di n asterischi. **Suggerimento.** Definite in modo induttivo l'operazione di stampa di n asterischi: per n=0 fate nulla, per n>0 stampate n-1 asterischi e alla fine stampatene ancora uno. **Esempio.** Stampate una linea di 10 asterischi.

Tut04.2 Scrivete una funzione ricorsiva

```
void stamparettangolo(int n, int m){…};
```

che preso in input due numero naturali n>=0, m>=0 stampi un rettangolo di n righe di m di asterischi ciascuna. **Suggerimento.** Riutilizzate la funzione che stampa la linea di asterischi. Definite in modo induttivo l'operazione di stampa di n linee tutte uguali, di m asterischi ciascuna. Per n=0 fate nulla, per n>0 stampate (n-1) linee di m asterischi, quindi andate a capo (se no non funziona!), infine stampate ancora una riga di m asterischi. **Esempio.** Stampate un quadrato di 10x10 asterischi.

Tut04.3 Scrivete una funzione ricorsiva

```
void stampatriangolo(int n){…};
```

che preso in input un numero naturale n>=0 stampi un triangolo rettangolo isoscele di asterischi con entrambi i cateti formato da n asterischi, e che inizi con un asterisco. Per n=3 dovete ottenere la seguente figura:

```
*
**
***
```

Suggerimento. Riutilizzate la funzione che stampa la linea di asterischi. Definite in modo induttivo la stampa del triangolo come segue. Se n=0 stampate nulla, se n>0 stampate un triangolo rettangolo di lato (n-1), quindi andate a capo (se no non funziona!), infine stampate ancora una riga di n asterischi. **Esempio.** Stampate il triangolo di lato 3.

Tut04.4 Scrivete una funzione ricorsiva

```
void stampabinario(int n)
```

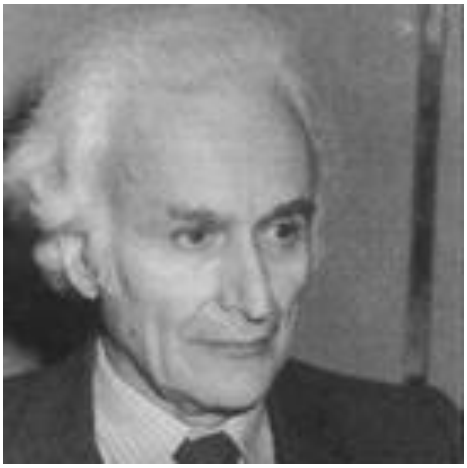
che stampi il corrispondente in base 2 di un numero decimale n>0, e nulla se n=0. **Suggerimento.** Definite in modo induttivo le operazioni di stampa. Se n=0 non stampate nulla. Se n>0 stampate prima la forma binaria di n/2, poi stampate n%2. **Esempio.** Stampate la forma binaria di 13: otterrete 1101.

Tutorato 04 – cenni di ricorsione
Parte 2. Soluzioni

```
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

/* Tut04.1. Linea di asterischi */
void stampalinea(int n)  //Sia n>=0
{if (n>0)
  {stampalinea(n-1);cout << '*';}
 /*altrimenti n=0 e faccio nulla*/
}

/* Tut04.2. rettangolo di asterischi */
void stamparettangolo(int n, int m)  //Siano n,m>=0
{if (n>0)
  {stamparettangolo(n-1,m);stampalinea(m);cout << endl;}
 /*altrimenti n=0 e faccio nulla*/
}

/* Tut04.3. triangolo di asterischi */
void stampatriangolo(int n)  //Sia n>=0
{if (n>0)
  {stampatriangolo(n-1);stampalinea(n);cout << endl;}
 /*altrimenti n=0 e faccio nulla*/
}


/* Tut04.4. versione binaria di un intero positivo */
void stampabinario(int n)  //Sia n>=0
{if (n>0)
  {stampabinario(n/2);cout << n%2;}
 /*altrimenti n=0 e faccio nulla*/
}

int main (){cout << "TEST stampalinea(10)" << endl;
 stampalinea(10); cout<<endl;system("PAUSE");system("CLS");

 cout << "TEST stamparettangolo(10,10)" << endl;
 stamparettangolo(10,10);
 cout<<endl;system("PAUSE");system("CLS");

 cout << "TEST stampatriangolo(10)" << endl;
 stampatriangolo(10);
 cout<<endl;system("PAUSE");system("CLS");

 cout << "TEST stampabinario(13)" << endl;
 stampabinario(13);
 cout<<endl;system("PAUSE");system("CLS");}
```

Settimana 05 – Ricorsione e Cicli While "con un while si può svolgere qualunque calcolo"

1. **Lezione 09: ricorsione e cicli while**
2. **Lezione 10: cicli while annidati**
3. **Tutorato 05: ricorsione e cicli while.**



Corrado Böhm. Dimostra nel 1966 il **Teorema di Böhm-Jacopini**: in linea di principio, le istruzioni while e if bastano a svolgere qualsiasi calcolo, perché *«ogni funzione calcolabile secondo la definizione di Turing è definibile a partire da cicli while, if e sequenze di istruzioni elementari».*

Lezione 09 – Ciclo While
Parte 1. Esempi (Cap. 6.1-6.4 testo)

Lezione 09. Prima ora. In questa lezione, parleremo del ***ciclo while***. Un ciclo è un modo per far ripetere a un computer ***la stessa operazione molte volte***. La ricorsione ottiene lo stesso effetto fornendo una definizione induttiva dell'operazione da svolgere. Un ciclo, invece, ***precisa in dettaglio quale azione ripetere e fino a quando ripeterla***. Un ciclo è un modo diretto e esplicito di programmare un computer, mentre la ricorsione è un modo indiretto e implicito.

/* Lezione09. Ciclo while: Capitoli 6.1-6.4 del libro di testo. Gli esempi del libro di testo.

while (condizione)
{istruzioni} //<- questa parte è detta CORPO del while

Un while ripete le istruzioni del corpo finche' la condizione è vera. Ripetere le istruzioni puo' cambiare il valore delle variabili nella condizione e rendere la condizione falsa: se e quando questo succede il while finisce, altrimenti il while non finisce mai. */

```
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

void countdown (int n) //Sia n>=0
 {
  while (n > 0)
   {
    cout << n << endl;
    n = n-1;
   }
//continuando a sottrarre 1 la variabile n diventa 0
//allora il while finisce e viene eseguita la riga dopo:
  cout << "Blastoff!" << endl;
 }
```

/*Per capire in dettaglio il funzionamento del countdown ne scriviamo una versione verbosa, "tracciando" l'esecuzione. "Tracciare" significa stampare, prima di eseguire ogni istruzione, quale istruzione stiamo per eseguire e con quali valori delle

variabili. "Tracciare" è molto utile per capire il funzionamento di un programma e rimuovere eventuali errori, ma rende il programma troppo verboso. Alla fine dobbiamo togliere tutte le istruzioni di "traccia", altrimenti il programma è inservibile.
Per aiutare la lettura scriviamo le "tracce" in rosso. */

```
void countdown_verboso (int n) //Sia n>=0
 {cout<<"-----------------------------------------------"<<endl;
  cout << "Sto per iniziare il while. n vale " << n << endl;
  while (n > 0)
   {cout<<"-----------------------------------------------"<<endl;
    cout<<"Sono all'inizio del corpo del while. Vale "
     << n << ">0. Stampo: " << endl;
    cout << n << endl;
    cout<<"Sono alla fine del corpo del while. Eseguo n=n-1 "
        <<endl;
    n = n-1;
    cout<<"-----------------------------------------------"<<endl;
   }
  cout<<"Ho appena terminato il while. Non vale "
   << n << ">0. Stampo: " << endl;
  cout << "Blastoff!" << endl;
 }
```

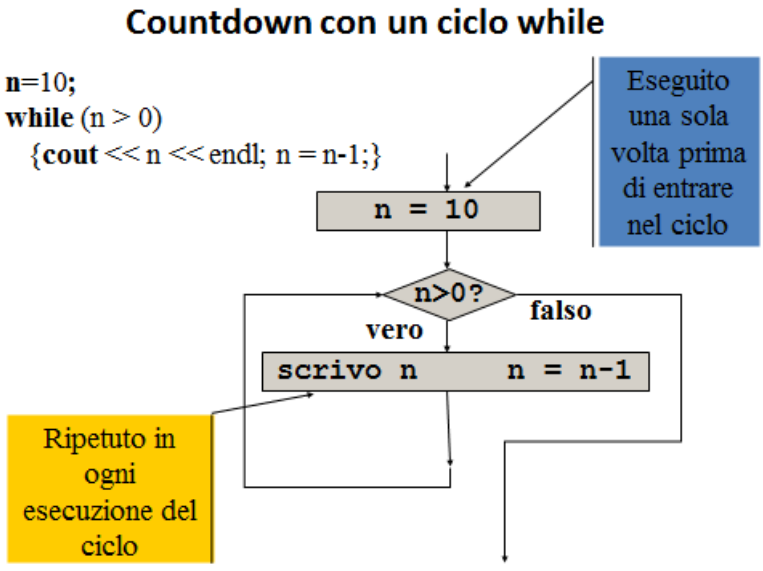
/* La "traccia" dell'esecuzione del countdown ci spiega cosa succede ad ogni passo del calcolo. Eseguendo countdown_verboso(3) otteniamo la seguente schermata:

```
-----------------------------------------------
Sto per iniziare il while. n vale 3
-----------------------------------------------
Sono all'inizio del corpo del while. Vale 3>0. Stampo:
3
Sono alla fine del corpo del while. Eseguo n=n-1
-----------------------------------------------
-----------------------------------------------
Sono all'inizio del corpo del while. Vale 2>0. Stampo:
2
Sono alla fine del corpo del while. Eseguo n=n-1
-----------------------------------------------
-----------------------------------------------
Sono all'inizio del corpo del while. Vale 1>0. Stampo:
1
Sono alla fine del corpo del while. Eseguo n=n-1
-----------------------------------------------
Ho appena terminato il while. Non vale 0>0. Stampo:
Blastoff
```

*/

```
/* Un while consente in linea di principio di scrivere qualunque
programma (Teorema di Bohm-Jacopini, 1966). Per il Teorema di
Indecidibilita' di Turing del 1937, ne segue che non esiste un
metodo meccanico per decidere, fissati dei valori per le
variabili, se un while termina per tutti i possibili valori
iniziali oppure no. Questo vale anche per while molto semplici.
 Un esempio: si ignora se il seguente while termini per ogni n
intero positivo oppure no (questo problema è noto come congettura
di Collatz). */

void StampaCollatzSequence (int n) //Sia n>0
{
 while (n != 1)
  {
   cout << n << endl;
   if (n%2 == 0) // n è pari
    {n = n / 2;}
   else // n è dispari
    {n = 3*n + 1;} //qui dovete scrivere 3*n e non 3n
  }
//se e quando n raggiunge 1 il while finisce.
 cout << 1 << endl;
}

/* La sequenza stampata dal while qui sopra viene detta sequenza
di Collatz. Provate a calcolare StampaCollatzSequence(n) per n=1,
10, 100, 1000: troverete sempre sequenze terminanti, ma non si sa
se questo vale per tutti gli n. */


int main()
{int n; double x;
 cout << "TEST countdown(10)" << endl;
 countdown(10);
 system("pause");system("CLS");

 cout << "TEST countdown(3)" << endl;
 countdown_verboso(3);
 system("pause");system("CLS");

 n=1;    cout << "Stampa sequenza Collatz per n=" << n << endl;
 StampaCollatzSequence (n);
 n=10;   cout << "Stampa sequenza Collatz per n=" << n << endl;
 StampaCollatzSequence (n);
 n=100;  cout << "Stampa sequenza Collatz per n=" << n << endl;
 StampaCollatzSequence (n);
 n=1000; cout << "Stampa sequenza Collatz per n=" << n << endl;
```

```
StampaCollatzSequence (n);
system("pause");system("CLS");

cout << "Tabella logaritmi in base e degli interi < 10" << endl;
x = 1.0;
  while (x < 10.0)
   {
   cout << x << "\t" << log(x) << "\n";
   x = x + 1.0;
  }
system("pause");system("CLS");

cout << "Tabella logaritmi in base 2 degli interi < 10" << endl;
x = 1.0;
  while (x < 10.0)
   {
   cout << x << "\t" << log(x)/log(2.) << "\n";
   x = x + 1.0;
  }
system("pause");system("CLS");}
```



Lezione 09 - Ciclo while
Parte 2. Esercizi

Lezione 09. Seconda ora. Tutti i prossimi esercizi richiedono un ciclo while.

9.1. Stampate dal **main** una linea di 10 asterischi usando un ciclo **while**. **Suggerimento.** Modificate il ciclo **while** per la funzione countdown.

9.2. Modificando l'esercizio precedente, e usando un ciclo **while**, scrivete una funzione

```
void stampalinea(int n){…}
```

che prende un intero n>=0 e stampa una linea di n asterischi.

9.3. Stampate dal **main** una tabella di due colonne, la prima colonna con gli interi tra 0 e 100, la seconda con la potenza di due corrispondente (il valore 2^{100} viene stampato in notazione esponenziale). Usate un ciclo **while**. Modificate l'esercizio sulla tabella dei logaritmi.

9.4. Modificando l'esercizio precedente, e sempre usando un ciclo **while**, scrivete una funzione

```
void TabellaPotenzeDue(int n)
```

che prende un intero n≥0 e stampa una tabella a due colonne con le potenze di due tra 0 e n compresi.

Lezione 09 - Ciclo while
Parte 3. Soluzioni

```
// Lezione09. Soluzioni
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

void stampalinea(int n)
{int i=1;
 while (i<=n)
  {
   cout << '*';
   i=i+1;
   }
}

void tabellapotenze(int n)
{int i=0;
 while (i<=n)
  {
   cout << "pow(2," << i << ")=" << "\t" << pow(2.,i) << endl;
   i=i+1;
  }
}

int main()
{int n; double x;

 cout << "TEST ciclo di stampa di 10 asterischi" << endl;
 int i=1;
 while (i<=10)
  {cout << '*';i=i+1;}
 cout << endl; system("pause");system("CLS");

 cout << "TEST funzione di stampa linea di asterischi" << endl;
 n=1;    cout << "Stampa n=" << n << " asterischi" << endl;
 stampalinea (n); cout << endl;
 n=10;   cout << "Stampa n=" << n << " asterischi" << endl;
 stampalinea (n); cout << endl;
 system("pause");system("CLS");

 cout << "Tabella potenze di 2 fino a 10" << endl;
 x = 1.0;
```

```
   while (x <= 10.0)
    {
    cout << x << "\t" << pow(2.,x) << "\n";
    x = x + 1.0;
   }
 system("pause");system("CLS");

 cout << "TEST funzione di stampa tabella potenze di 2" << endl;
 n=10; cout << "tabellapotenze(" << n << ")" << endl;
 tabellapotenze(n);
 n=100; cout << "tabellapotenze(" << n << ")" << endl;
 tabellapotenze(n);
 system("pause");system("CLS");
}
```

Lezione 10. Cicli While annidati
Parte 1. Esempi (Cap. 6.5-6.10 libro testo)

In questa lezione vedremo dei cicli while che fanno ripetere una funzione, definita a sua volta con un ciclo while. Questi cicli while vengono detti ***"annidati"***, che significa "scritti uno dentro l'altro". L'esempio, tratto dal libro di testo, è una funzione che stampa la ***tavola pitagorica*** di dimensioni nxm.

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

```
// Lezione10. PRIMA MEZZ'ORA. Cap. 6.5-6.10
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

/* ESEMPIO 1 - LEZIONE 10. funzione che generalizza il ciclo che
stampa 6 multipli di 2 a un ciclo che stampa 6 multipli di n */
void printMultiples (int n)
{
 int i = 1;
 while (i <= 6)
  {
   cout << n*i << "\t";
   i = i + 1;
  }
 cout << endl;
}
```

```
/* ESEMPIO 2 - LEZIONE 10. funzione che stampa la lista dei primi
6 multipli dei valori da 1 fino a n */
void printMultTable (int high)
 {
  int i = 1;
  while (i <= high)
  {
   printMultiples (i);
   i = i + 1;
  }
 }

/* ESEMPIO 3 - LEZIONE 10. Seconda versione printMultiples:
ridefiniamo printMultiples(n,high) in modo che possa stampare la
tabellina di n fino a high: n, n*2, n*3, …, n*high. A partire da
questa nuova funzione definiamo printMultTable2(n) che stampa le
tabelline nxn. */

void printMultiples (int n, int high)
/* posso tenere lo stesso nome printMultiples perche' il numero di
argomenti è diverso e non c'è rischio di confusione */
{
 int i = 1;
 while (i <= high)
  {cout << n*i << "\t";i = i + 1;}
 cout << endl;
}

/* ESEMPIO 3 - LEZIONE 10 (Continua e finisce). Con la nuova
versione di printMultiples definiamo printMultTable2(n) che stampa
le tabelline nxn. */

void printMultTable2 (int high)
{int i = 1;
 while (i <= high)
  {printMultiples (i, high);i = i + 1;}}


int main()
{int i, n;
 cout << "Ciclo che stampa 6 multipli di 2 in riga" << endl;
 i = 1;
 while (i <= 6)
  {cout << 2*i << "\t"; i = i + 1;}
 cout << endl;
```

```
 system("pause");system("CLS");

/* ESEMPIO 1 - LEZIONE 10. TEST funzione che stampa i primi 6
multipli di n */
 cout << "ESEMPIO 1. TEST funzione che stampa multipli di n" <<
endl;
 n = 1; cout << " Stampa 6 Multipli di n=" << n << endl;
 printMultiples (n);
 n = 2; cout << " Stampa 6 Multipli di n=" << n << endl;
 printMultiples (n);
 n = 3; cout << " Stampa 6 Multipli di n=" << n << endl;
 printMultiples (n);
 system("pause");system("CLS");

 cout << "Ciclo che stampa una tabella delle moltiplicazioni 6x6"
<< endl;
 i = 1;
 while (i <= 6)
  {
   printMultiples (i);
   i = i + 1;
  }
 system("pause");system("CLS");

 /*ESEMPIO 2-LEZIONE 10. TEST funzione che stampa tabelline nx6*/
cout << "ESEMPIO 2. TEST funzione che stampa tabelline nx6"
  << endl;
 n = 10; cout << " Stampa 6 Multipli di n da 1 fino a " << n <<
endl;
 printMultTable (n);
 system("pause");system("CLS");

 /* ESEMPIO 3 - LEZIONE 10. Saltiamo il test di
printMultiples(n,high), che stampa la tabellina di n da 1 fino a
high. Eseguiamo un test su printMultTable(n) che stampa le
tabelline nxn*/
 cout << " ESEMPIO 3. TEST funzione che stampa tabelline nxn"
 << endl;
 n = 10; cout << " Stampa tabellina moltiplicazioni " << n << "x"
 << n << endl;
 printMultTable2 (n); system("pause");system("CLS");}
```

Lezione 10 - Cicli while annidati
Parte 2. Esercizi

Adattate le soluzioni viste all'inizio della lezione per risolvere gli esercizi seguenti, in cui un ciclo while fa ripetere una funzione definita con un ciclo while. Ricopiate nel vostro file la funzione della lezione precedente: **void** stampalinea(**int** n){…}, che prende un intero n≥0 e stampa una linea di n asterischi:
****

10.1. Usando la funzione stampalinea(n), fate stampare nel **main** un **rettangolo** di 10x10 asterischi usando un ciclo **while:**
****
****
****
****
****
****
****
****
****
****

10.2. Modificando l'esercizio precedente, scrivete una funzione
void stamparettangolo(**int** n, **int** m){…}
che prende degli interi n,m>=0 e stampa un **rettangolo** di nxm asterischi, usando un ciclo **while.**

10.3. Usando la funzione stampalinea(n), fate stampare nel **main** un **triangolo** rettangolo isoscele di asterischi di lato 10 usando un ciclo **while.**
*****
******
****
****
****
****
****
****
****
****

10.4. Modificando l'esercizio precedente, scrivete una funzione
void stampatriangolo(**int** n){…}
che prende un intero n≥0 e stampa un **triangolo** rettangolo isoscele di asterischi di lato n, usando un ciclo **while.**

Lezione 10 - Cicli while annidati
Parte 3. Soluzioni

```
// Lezione10. Soluzioni
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

void stampalinea(int n)
{int i=1;
 while (i<=n)
  {
   cout << '*';
   i=i+1;
   }
}

void stamparettangolo(int n, int m)
{int i=1;
 while (i<=n)
  {
   stampalinea(m);cout << endl;
   i=i+1;
  }
}

void stampatriangolo(int n)
{int i=1;
 while (i<=n)
  {
   stampalinea(i); cout << endl;
   i=i+1;
  }
}

int main()
{int i;

 cout << "TEST ciclo di stampa di 10x10 asterischi" << endl;
 i=1;
 while (i<=10)
  {stampalinea(10);cout << endl;i=i+1;}
 system("pause");system("CLS");
```

```
 cout << "TEST funzione di stampa di nxn asterischi" << endl;
 i=1;cout << "stamparettangolo(" << i << "," << i << ")" << endl;
 stamparettangolo(i,i);
 i=2;cout << "stamparettangolo(" << i << "," << i << ")" << endl;
 stamparettangolo(i,i);
 i=3;cout << "stamparettangolo(" << i << "," << i << ")" << endl;
 stamparettangolo(i,i);
 i=4;cout << "stamparettangolo(" << i << "," << i << ")" << endl;
 stamparettangolo(i,i);
 system("pause");system("CLS");

 cout << "TEST ciclo di stampa di un triangolo di lato 10" <<
endl;
 i=1;
 while (i<=10)
  {stampalinea(i);cout << endl;i=i+1;}
 system("pause");system("CLS");

 cout << "TEST funzione di stampa di triangolo di lato n" << endl;
 i=1;cout << "stampatriangolo(" << i << ")" << endl;
 stampatriangolo(i);
 i=2;cout << "stampatriangolo(" << i << ")" << endl;
 stampatriangolo(i);
 i=3;cout << "stampatriangolo(" << i << ")" << endl;
 stampatriangolo(i);
 i=4;cout << "stampatriangolo(" << i << ")" << endl;
 stampatriangolo(i);
 system("pause");system("CLS");
}
```

Lezione 10 – Parte 4. Ripasso della Ricorsione e confronto con i cicli While

Vogliamo ripassare la ricorsione e confrontarla con i cicli while. Vedremo come le definizioni ricorsive richiedono una definizione induttiva rigorosa di ciò che vogliamo calcolare, ma hanno il vantaggio di far produrre al compilatore un programma sicuro. Invece, i cicli while non hanno bisogno di una definizione induttiva, quindi sono più facili da scrivere, ma ci richiedono di spiegare in dettaglio come ottenere il risultato voluto. Come conseguenza, un ciclo while ha un maggior rischio di errori. Useremo i cicli while nella maggioranza dei casi: il prezzo da pagare è fare molta attenzione a ciò che scriviamo.

Per studiare in dettaglio un programma ricorsivo, tracciamo le esecuzioni degli esercizi 8.2 e 8.3 della lezione 08, per vedere come il compilatore C++ utilizza le definizioni delle funzioni ricorsive. ***Solo per completezza***, nelle dispense includiamo anche le tracce delle esecuzioni di 8.4 e 8.5.

1. All'inizio di ogni definizione di funzione facciamo stampare il valore da calcolare. Per 8.1 (funzione potenza(a,b)), se dobbiamo calcolare potenza(10,3) scriviamo

   calcolo potenza(10,3)

2. Prima di ogni **return** facciamo stampare la formula usata e il valore trovato. Per 8.2 (funzione potenza(a,b)), se dobbiamo calcolare potenza(10,3) scriviamo la formula

   potenza(10,3)=potenza(10,2)*10=1000

mentre se dobbiamo calcolare potenza(10,0) scriviamo solo

   potenza(10,0)=1

Come risultato, otteniamo una lista di TUTTE le definizione usate dal compilatore per trasformare la definizione induttiva in un programma. Questa lista ci fa capire come vengono calcolate le funzioni ricorsive in un caso semplice.

Nella prossima lezione risolveremo gli stessi esercizi usando cicli while. I programmi scritti da noi produrranno calcoli più semplici e veloci rispetto a quelli scritti dal compilatore a partire dalle definizioni ricorsive. Tuttavia, come abbiamo anticipato, sarà anche più difficile controllare se il programma che abbiamo scritto è corretto.

Traccia di 8.2. La funzione è

```
double potenza(double a, int b)
{if (b==0){return 1;} else /*b>0*/ {return potenza(a,b-1)*a;} };
```

Eseguendo una versione con traccia otteniamo a video:

```
TRACCIA CALCOLO potenza(a,b)
calcolo potenza(10,3)
calcolo potenza(10,2)
calcolo potenza(10,1)
calcolo potenza(10,0)
potenza(10,0)=1
potenza(10,1)=potenza(10,0)*10=10
potenza(10,2)=potenza(10,1)*10=100
potenza(10,3)=potenza(10,2)*10=1000
RISULTATO potenza(10,3)=1000
```

Questo ci dice che il calcolo di potenza(10,3) si riconduce al calcolo di potenza(10,2), quindi al calcolo di potenza(10,1) e al calcolo di potenza(10,0). Quest'ultimo vale 1, l'equazione potenza(10,1) = potenza(10,0) * 10 = 10 ci consente di calcolare potenza(10,1), da quest'ultimo calcoliamo potenza(10,2) = potenza(10,1) * 10 = 100 e alla fine potenza(10,3) = potenza(10,2) * 10 = 1000.

Analogamente possiamo comprendere la soluzione di 8.3 studiando la sua traccia di esecuzione.

8.3. Calcolo di SD(n) = 1 + 3 + 5 + … + (2n-1)

```
int SD(int n)  //Sia n≥0
{if (n==0){return 0;} else /*n>0*/ {return SD(n-1)+(2*n-1);}};
```

```
TRACCIA CALCOLO SD(n)
calcolo SD(4)
calcolo SD(3)
calcolo SD(2)
calcolo SD(1)
calcolo SD(0)
SD(0)=0
SD(1)=SD(0)+1=1
SD(2)=SD(1)+3=4
SD(3)=SD(2)+5=9
SD(4)=SD(3)+7=16
RISULTATO SD(4)=16
```

Solo per completezza, includiamo qui la traccia anche degli esercizi 8.4 e 8.5. Non abbiamo svolto a lezione questa parte.

8.4. Calcolo di log(a,b) = logaritmo in base a di b, parte intera.

```
int log(int a, int b) //Siano a≥2, b≥1
{if ((b/a)==0)
  {return 0;}
 else /*(b/a)≥ 1*/
 {return 1+log(a,b/a);}}
```

```
TRACCIA CALCOLO log(a,b)

calcolo log(3,1000)
calcolo log(3,333)
calcolo log(3,111)
calcolo log(3,37)
calcolo log(3,12)
calcolo log(3,4)
calcolo log(3,1)
log(3,1)=0
log(3,4)=log(3,1)+1=1
log(3,12)=log(3,4)+1=2
log(3,37)=log(3,12)+1=3
log(3,111)=log(3,37)+1=4
log(3,333)=log(3,111)+1=5
log(3,1000)=log(3,333)+1=6
RISULTATO log(3,1000)=6
```

/* 8.5. MCD(a,b)*/

```
int MCD(int a, int b) //Siano a>0, b≥0
{if (a==0){return b;} else /*a>0*/ {return MCD(b%a,a);}};
```

```
TRACCIA CALCOLO MCD(a,b)
calcolo MCD(144,111)
calcolo MCD(111,144)
calcolo MCD(33,111)
calcolo MCD(12,33)
calcolo MCD(9,12)
calcolo MCD(3,9)
calcolo MCD(0,3)
MCD(0,3)=3
MCD(3,9)=MCD(0,3)=3
MCD(9,12)=MCD(3,9)=3
MCD(12,33)=MCD(9,12)=3
MCD(33,111)=MCD(12,33)=3
MCD(111,144)=MCD(33,111)=3
MCD(144,111)=MCD(111,144)=3
RISULTATO MCD(144,111)=3
```

Di seguito inseriamo le soluzioni di 8.2-8.5 insieme alle istruzioni **cout** che generano le tracce delle loro esecuzioni.

```
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

/* Traccia esecuzione es. 8.2. potenza */
double potenza(double a, int b) //Sia b≥0
{cout << "calcolo potenza(" << a << "," << b << ")" << endl;
 if (b==0)
  {double result = 1;
   //stampo risultato e formula usata
   cout << "potenza(" << a << "," << b << ")=" << result << endl;
   return result;}
 else //b>0
  {double result = potenza(a,b-1)*a;
   //stampo risultato e formula usata
   cout << "potenza(" << a << "," << b << ")="
   << "=potenza(" << a << "," << b-1 << ")*"<< a <<"=" << result
<< endl;
   return result;}
};

/* 8.3. Somma dei primi n dispari */
int SD(int n) //Sia n≥0
{cout << "calcolo SD(" << n << ")" << endl;
 if (n==0)
  {int result = 0;
   //stampo risultato e formula usata
   cout << "SD(" << n << ")=" << result << endl;
   return result;}
 else //n>0
  {int result = SD(n-1)+(2*n-1); /*non scrivete 2n-1 ma 2*n-1*/
   //stampo risultato e formula usata
   cout << "SD(" << n << ")="
   << "SD(" << n-1 << ")+" << 2*n-1 << "=" << result << endl;
   return result;}
};

/* 8.4. parte intera di log(a,b) */
int log(int a, int b) //Siano a≥2, b≥1
{cout << "calcolo log(" << a << "," << b << ")" << endl;
 if ((b/a)==0)
```

```
  {int result = 0;
   //stampo risultato e formula usata
   cout << "log(" << a << "," << b << ")=" << result << endl;
   return result;}
 else //(b/a)≥1
  {int result = 1+log(a,b/a);
   //stampo risultato e formula usata
   cout << "log(" << a << "," << b << ")=" <<
   "log(" << a << "," << b/a << ")+1=" << result << endl;
   return result;}
};

/* 8.5. MCD(a,b) */
int MCD(int a, int b) //Siano a>0, b≥0
{cout << "calcolo MCD(" << a << "," << b << ")" << endl;
 if (a==0)
  {int result = b;
   //stampo risultato e formula usata
   cout << "MCD(" << a << "," << b << ")=" << result << endl;
   return result;}
 else //a>0
  {int result = MCD(b%a,a);
   //stampo risultato e formula usata
   cout << "MCD(" << a << "," << b << ")=" <<
   "MCD(" << b%a << "," << a << ")=" << result << endl;
   return result;}
};

/* 8.6. mcm(a,b) */
int mcm(int a, int b) //Siano a>0, b>0
{cout << "calcolo mcd(" << a << "," << b << ")" << endl;
 int result = (a*b)/MCD(a,b);
 //stampo risultato e formula usata
 cout << "mcd(" << a << "," << b << ")=" << a << "*" << b <<
 "/MCD(" << a << "," << b << ")=" << result << endl;
 return result;};


int main (){ double x; int a,b,n;

 cout << "TRACCIA CALCOLO potenza(a,b)" << endl;
 x=10;n=3;
 cout << "RISULTATO potenza(" << x << "," << n << ")="
 << potenza(x,n) << endl;
 system("PAUSE");system("CLS");
```

```
 cout << "TRACCIA CALCOLO SD(n)" << endl;
 n=4;
 cout << "RISULTATO SD(" << n << ")=" << SD(n) << endl;
 system("PAUSE");system("CLS");

 cout << "TRACCIA CALCOLO log(a,b)" << endl;
 a=3;b=1000;
 cout << "RISULTATO log(" << a << "," << b << ")="
 << log(a,b) << endl;
 system("PAUSE");system("CLS");

 cout << "TRACCIA CALCOLO MCD(a,b)" << endl;
 a=144;b=111;
 cout << "RISULTATO MCD(" << a << "," << b << ")="
 << MCD(a,b) << endl;
 system("PAUSE");system("CLS");

 cout << "TRACCIA CALCOLO mcm(a,b)" << endl;
 a=10;b=15;cout << "RISULTATO mcm(" << a << "," << b << ")="
 << mcm(a,b) << endl;
 system("PAUSE");system("CLS");
}
```

Tutorato 05 – ricorsione e cicli while
Parte 1. Esercizi proposti

Negli esercizi seguenti vi chiediamo di scrivere una funzione con una definizione ricorsiva, e poi, come confronto, la stessa funzione usando un ciclo while. Nel caso della definizione ricorsiva, voi dovete prima trovare una definizione del risultato per induzione su n oppure su m, e poi ricopiarla in C++.

Attenzione. Gli esercizi 5.3 e 5.5 richiedono funzioni con un valore di ritorno. Quando ricopiate una definizione induttiva in C++, se volete un valore di ritorno v ricordatevi di aggiungere un **return** davanti al risultato in ognuno dei casi della definizione ricorsiva. Evitate di inserire un return dentro un while: il return fa uscire prematuramente dal ciclo anche se la condizione è ancora vera. Uscire prematuramente da un ciclo while va fatto solo in casi particolari, che vedremo più avanti.

Tut05.1 Scrivere una funzione ricorsiva

void stampapariREC(**int** n)

che stampi tutti i numeri pari in [2,n] a partire dall'ultimo. Analogamente scrivere una funzione ricorsiva

void stampadispariREC(**int** n)

che stampi tutti i numeri dispari in [1,n] a partire dall'ultimo. **Esempi:** se n=10 allora stampapariREC(10) stampa 10, 8, 6, 4, 2, mentre stampadispariREC(10) stampa 9, 7, 5, 3, 1. **Suggerimento.** Usate induzione su n e distinguete i casi n pari e n dispari.

Tut05.2 Scrivere una funzione utilizzando un ciclo **while**

void stampapari(**int** n)

che stampi tutti i numeri pari in [2,n] a partire dal primo. Analogamente scrivere una funzione utilizzando un ciclo **while**

void stampadispari(**int** n)

che stampi tutti i numeri dispari in [1,n] a partire dal primo. **Esempi:** vedi sopra.

Tut05.3 (con valore di ritorno) Scrivere una funzione **ricorsiva**

int sommamultipliREC(**int** n, **int** m)

che calcoli la somma di tutti i multipli di m che sono ≤ n. **Esempio:** sommamultipliREC(10,2) = 2 + 4 + 6 + 8 + 10 = 30. **Suggerimento.** Usate induzione su n e distinguete i casi in cui m divide n e quelli in cui m non divide n.

Tut05.4 (con valore di ritorno) Scrivere una funzione utilizzando un ciclo **while**

int sommamultipli(**int** n, **int** m)

che calcoli la somma di tutti i multipli di m minori o uguali ad n. **Esempio:** vedi sopra. **Suggerimento.** Usate un indice k che assume nell'ordine tutti i valori m, 2m, 3m, … che sono ≤ n. Sommate

tutti i valori di k dentro una variabile sum, e alla fine restituite sum.

Tut05.5 (con valore di ritorno) Scrivere una funzione **ricorsiva**

int sommadivisoriREC(**int** n, **int** m)

che calcoli la somma dei divisori di n minori o uguali ad m. **Esempio:** sommadivisoriREC(10,10) = 1 + 2 + 5 + 10 = 18. **Suggerimento.** Usate induzione su m e distinguete i casi in cui m divide n e quelli in cui m non divide n.

Tut05.6 (con valore di ritorno) Scrivere una funzione utilizzando un ciclo **while**

int sommadivisori(**int** n, **int** m)

che calcoli la somma dei divisori di n che sono ≤ m.

Tutorato 05 – ricorsione e cicli while
Parte 2. Soluzioni

Tut05.1 Scrivere una funzione ricorsiva

void stampapariREC(**int** n)

che stampi tutti i numeri pari in [2,n]. Analogamente scrivere una funzione ricorsiva

void stampadispariREC(**int** n)

che stampi tutti i numeri dispari in [1,n].

```
void stampapariREC(int n){
     if (n==0) cout << "Numeri Pari=";
     else if (n%2==0) {
           cout << n<<";";
           stampapariREC(n-2);
     }
     else stampapariREC(n-1);
}

void stampadispariREC(int n){
     if (n<0) cout <<"Numeri Dispari=";
     else if (n%2==1) {
           cout << n <<";";
           stampadispariREC(n-2);
     }
     else stampadispariREC(n-1);
}
```

Tut05.2 Scrivere una funzione utilizzando un ciclo **while**

void stampapari(**int** n)

che stampi tutti i numeri pari in [2,n]. Analogamente scrivere una funzione utilizzando un ciclo **while**

void stampadispari(**int** n)

che stampi tutti i numeri dispari in [1,n].

```
void stampapari(int n){
     int k=2;
     while(k<=n){
        cout << k <<";";
        k=k+2;
     }
     cout<<endl;
}

void stampadispari(int n){
     int k=1;
     while(k<=n){
        cout << k <<";";
        k=k+2;
     }
     cout<<endl;
}
```

Tut05.3 Scrivere una funzione ricorsiva

```
int sommamultipliREC(int n, int m)
```

che calcoli la somma di tutti i multipli di m minori o uguali ad n.

```
int sommamultipliREC(int n, int m){
    if (n < m || m==0) return 0;
    else if (n % m == 0) return n + sommamultipliREC(n-m,m);
    else return sommamultipliREC(n-1,m);
}
```

Tut05.4 Scrivere una funzione utilizzando un ciclo **while**

```
int sommamultipli(int n, int m)
```

che calcoli la somma di tutti i multipli di m minori o uguali ad n.

```
int sommamultipli(int n, int m){
     int k = m;
     int sum = 0;
     while(k<=n){
        sum = sum + k;
        k=k+m;
     }
     return sum;
}
```

Tut05.5 Scrivere una funzione ricorsiva

```
int sommadivisoriREC(int n, int m)
```

che calcoli la somma dei divisori di n minori o uguali ad m.

```
int sommadivisoriREC(int n, int m){
    if (m == 0 || n==0 ) return 0;
    else if (n % m == 0) return m + sommadivisoriREC(n,m-1);
    else return sommadivisoriREC(n,m-1);
}
```

Tut05.6 Scrivere una funzione utilizzando un ciclo **while**

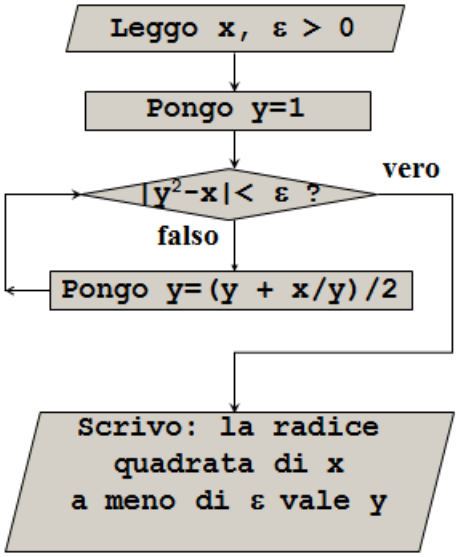
```
int sommadivisori(int n, int m)
```

che calcoli la somma dei divisori di n minori o uguali ad m.

```
int sommadivisori(int n, int m){
     int k = 1;
     int sum = 0;
     while(k<= m){
        if (n%k==0) sum = sum + k;
        k++;
     }
     return sum;}
```

Settimana 06 – Esercitazioni sui cicli while

1. **Lezione 11: esercitazioni sui cicli while**
2. **Lezione 12: esercitazioni sui cicli while**
3. **Tutorato 06: esercitazione sui cicli while.**



L'algoritmo detto "babilonese" per il calcolo della radice quadrata può essere realizzato con un ciclo while

Lezione 11 – Esercizi su cicli While

Parte 1. Esempi e Esercizi

In questa settimana vedremo esempi non banali di cicli while. Iniziamo la lezione vedendo un ***esempio*** non banale di uso di un while all'interno di un altro ciclo while. Il programma è descritto in dettaglio nell'Appendice 1 di queste Dispense, ma per ragioni di tempo ci limitiamo ad accennarlo. Usiamo un ciclo while per stampare dei cerchi colorati lungo una ***spirale di archimede*** di velocità di rotazione alpha. Un secondo ciclo while, posto all'esterno del primo, ridisegna più volte la spirale ***con una velocità di rotazione alpha via via maggiore***. L'effetto è una **animazione** di una bellezza e complessità sorprendenti. Questo esempio ha uno scopo puramente dimostrativo e non fa parte del programma di esame.

Ora recuperiamo gli esercizi già fatti con la ricorsione, ma per confronto proviamo a svolgerli usando un ciclo **while**, senza ricorsione. Di conseguenza, dovremo precisare in grande dettaglio come il computer deve svolgere il calcolo, anziché limitarci a definire il risultato per induzione, come si fa quando si programma in modo ricorsivo. Tutte le funzioni richieste hanno un valore di ritorno: devono restituire un valore e non stampare un valore. Ricordatevi di non inserire un return dentro un while: fa uscire prematuramente dal ciclo.

11.1. Scrivete una funzione con ciclo **while**

```
double potenza(double a, int b) {…};
```

che dati un reale a e un intero b>=0 restituisca il reale a^b. Non utilizzate la funzione pow già disponibile in libreria. **Suggerimento.** Usate una variabile reale result dove "accumulare" le potenze successive di a, fino ad arrivare alla potenza numero b. Inizializzate result al valore a^0=1, quindi usate un ciclo **while** per porre ripetutamente result=result*a. Per tener conto di quante volte avete moltiplicato per a, usate una variabile intera i inizializzata a 1, e ricordatevi di incrementarla di 1 ogni volta che moltiplicate per a. Uscite dal **while** quando i>b. A questo punto, ricordatevi di restituire il valore corrente di result come valore di ritorno. **Esempi.** Controllate che sia: 10^0=1, 10^1=10, 10^2=100, 10^3=1000.

11.2. Scrivete una funzione con ciclo **while**

```
int SD (int n) {…};
```

che prenda un intero n>=0 e restituisca come valore di ritorno l'intero (1+3+5+…+(2n-1)) (ovvero la somma dei primi n dispari).

Suggerimento. Adattate la soluzione per il ciclo **while** per la potenza, e usate una variabile result in cui accumulare la somma dei primi n dispari. La variabile result ora deve essere di tipo intero, è inizializzata a 0 e incrementata di +(2i-1) quando sommate il dispari numero i. La variabile i che fa da contatore è intera, parte da 1 e viene incrementata di 1 alla fine di ogni passo. Uscite dal **while** quando i>n. A questo punto, ricordatevi di restituire il valore di result come valore di ritorno. **Esempi.** Controllate che sia: SD(0)=0, SD(1)=1, SD(2)=4, SD(3)=9, SD(4)=16.

11.3. Scrivete una funzione con ciclo **while**

```
int log(int a, int b) {…};
```

che dati due interi a≥2, b≥1 restituisce $\log_a(b)$ arrotondato per difetto a un valore intero. Non utilizzate la funzione log già disponibile in libreria. **Suggerimento.** Contate quante volte è possibile eseguire la divisione arrotondata b |→ b/a mantenendo vera la condizione b≥a. Per farlo, usate una variabile result di tipo intero, inizializzata a 0 e incrementata di 1 ogni volta che rimpiazzate b con b/a. result è sia un accumulatore per il risultato che un contatore: in questo caso non serve un contatore i. L'assegnazione b=b/a, un decremento, tiene il posto dell'incremento usato in altri cicli. Uscite dal **while** quando b < a. A questo punto, ricordatevi di restituire il valore result come valore di ritorno. **Esempi.** Controllate che sia: log(3,1)=0, log(3,10)=2, log(3,100)=4, log(3,1000)=6.

11.4. ***(Massimo Comun Divisore).*** Scrivete una funzione con ciclo **while** **int** MCD(**int** a, **int** b) che calcoli il Massimo Comun Divisore di due interi a≥0, b>0. **Suggerimento.** Usate l'algoritmo di Euclide, definito come segue. Rimpiazzate ripetutamente la coppia di interi (a,b) con (b%a,a), finche' a>0, quando a=0 restituite b. Per farlo, evitate di scrivere a=b%a, questo cancella a prima che venga assegnato a b. Eseguite invece le assegnazioni: **int** v1 = b%a; **int** v2 = a; a = v1; b = v2; Quando a vale 0, uscite dal ciclo **while**. A questo punto, ricordatevi di restituire b come valore di ritorno. **Esempi.** Controllate che sia: MCD(0,11) = 11, MCD(13,18) = 1, MCD(144,111) = 3, MCD(144,96) = 48.

11.5. ***(Minimo comune multiplo).*** Scrivete una funzione mcm(**int** a, **int** b) che calcoli il minimo comune multiplo di due interi positivi a, b. La funzione non richiede cicli **while** ne' ricorsione. **Suggerimento.** Usate la funzione MCD definita in precedenza e la formula mcm(a,b) = (a*b)/MCD(a,b). **Esempi.** Controllate che sia: mcm(1,11) = 11, mcm(12,5) = 60, MCD(144,48) = 144, mcm(10,15)=30.

Lezione 11 – Esercizi su cicli While
Parte 2. Soluzioni

```
// Lezione11-Esempi di cicli while-Soluzioni
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

/* 11.1. potenza */
double potenza(double a, int b) //Sia b>=0
{int i=1;double result=1;
 while(i<=b)
  {result=result*a;
   i=i+1;}
 return result;};

/* 11.1. potenza con TRACCIA dell'esecuzione. Definiamo una
versione della funzione potenza in cui a ogni passo del calcolo
stampiamo i valori delle variabili. Possiamo renderci conto di
come il while aggiorna i valori di result e di i. */
double potenza_verbosa(double a, int b)
{int i = 1; double result = 1;
 cout << "----------------------" << endl;
 cout << "potenza(" << a << "," << b << ")=?" << endl;
 cout << "PRIMA DEL CICLO WHILE" << endl;
 cout << "result = " << result << " i = " << i << endl;
 while (i <= b)
  {
   result = result*a;
   i = i+1;
   cout << " DOPO UNA ESECUZIONE DEL CORPO DEL WHILE" << endl;
   cout << " result = " << result << " i = " << i << endl;
  }
 cout << "ALL'USCITA DEL WHILE" << endl;
 cout << "result = " << result << " i = " << i << endl;
 cout << "----------------------" << endl;
 return result;
};

/* 11.2. Somma dei primi n dispari */
int SD(int n) //Sia n>=0
{int i=1; int result=0;
 while(i<=n)
```

```
  {result=result+(2*i-1); //non scrivete (2i-1)
   i=i+1;}
 return result;};

/* 11.3. parte intera di log(a,b) */
int log(int a, int b) //Siano a>=2, b>=1
{int result=0;
 while(b>=a)
  {b=b/a;
   result=result+1;}
 return result;};

/* 11.4. MCD(a,b) */
int MCD(int a, int b) //Siano a>0, b≥0
{while(a>0)
 {int v1 = b%a;
  int v2 = a;
      a  = v1;
      b  = v2;}
 return b;};

/* 11.5. mcm(a,b) */
int mcm(int a, int b) //Siano a>0, b>0
{return (a*b)/MCD(a,b);};

int main (){double x;int a,b,n;

 cout << "TEST potenza(x,n)" << endl;
 x=10;n=0; cout << "potenza(" << x << "," << n << ")="
 << potenza(x,n) << endl;
 x=10;n=1; cout << "potenza(" << x << "," << n << ")="
 << potenza(x,n) << endl;
 x=10;n=2; cout << "potenza(" << x << "," << n << ")="
 << potenza(x,n) << endl;
 x=10;n=3; cout << "potenza(" << x << "," << n << ")="
 << potenza(x,n) << endl;
 system("PAUSE");system("CLS");

 cout << "TEST potenza_verbosa(x,n)" << endl;
 x=10;n=0; cout << "potenza(" << x << "," << n << ")="
 << potenza_verbosa (x,n) << endl;
 x=10;n=1; cout << "potenza(" << x << "," << n << ")="
 << potenza_verbosa(x,n) << endl;
 x=10;n=2; cout << "potenza(" << x << "," << n << ")="
 << potenza_verbosa (x,n) << endl;
```

```
x=10;n=3; cout << "potenza(" << x << "," << n << ")="
<< potenza_verbosa (x,n) << endl;
system("PAUSE");system("CLS");

cout << "TEST SD(n)" << endl;
n=0; cout << "SD(" << n << ")=" << SD(n) << endl;
n=1; cout << "SD(" << n << ")=" << SD(n) << endl;
n=2; cout << "SD(" << n << ")=" << SD(n) << endl;
n=3; cout << "SD(" << n << ")=" << SD(n) << endl;
n=4; cout << "SD(" << n << ")=" << SD(n) << endl;
system("PAUSE");system("CLS");

cout << "TEST log(a,b)" << endl;
a=3;b=1; cout << "log(" << a << "," << b << ")="
<< log(a,b) << endl;
a=3;b=10; cout << "log(" << a << "," << b << ")="
<< log(a,b) << endl;
a=3;b=100; cout << "log(" << a << "," << b << ")="
<< log(a,b) << endl;
a=3;b=1000; cout << "log(" << a << "," << b << ")="
<< log(a,b) << endl;
system("PAUSE");system("CLS");

cout << "TEST MCD(a,b)" << endl;
a=0;b=11;cout << "MCD(" << a << "," << b << ")="
<< MCD(a,b) << endl;
a=13;b=18;cout << "MCD(" << a << "," << b << ")="
<< MCD(a,b) << endl;
a=144;b=111;cout << "MCD(" << a << "," << b << ")="
<< MCD(a,b) << endl;
a=144;b=96;cout << "MCD(" << a << "," << b << ")="
<< MCD(a,b) << endl;
system("PAUSE");system("CLS");

cout << "TEST mcm(a,b)" << endl;
a=1;b=11;cout << "mcm(" << a << "," << b << ")="
<< mcm(a,b) << endl;
a=12;b=5;cout << "mcm(" << a << "," << b << ")="
<< mcm(a,b) << endl;
a=144;b=48;cout << "mcm(" << a << "," << b << ")="
<< mcm(a,b) << endl;
a=10;b=15;cout << "mcm(" << a << "," << b << ")="
<< mcm(a,b) << endl;
system("PAUSE");system("CLS");
}
```

Lezione 12 - Esercizi su cicli While

Parte 1. Esercizi

In questa lezione vediamo esercizi su i cicli while di difficoltà paragonabile a quella di un esercizio di esame. Cominciamo co qualche suggerimento generale.

1. I primi tre esercizi, da 12.1 a 12.3, si assomigliano: richiedono di calcolare una sommatoria $\Sigma_{1\le i\le n}\ a_i$ dei primi n valori di una successione $a_1,\dots,a_n$. Cambia soltanto la formula per calcolare a_i, e a volte l'indice i ha valore iniziale 0 anziché da 1. Dovete adattare la soluzione dell'esercizio 11.2 sulla somma dei primi n dispari.
2. Gli altri due esercizi, 12.4 e 12.5, sono abbastanza diversi. L'algoritmo per il calcolo della radice quadrata (es. 12.4) ***non ha bisogno di contatore***: non alcuna importanza quante volte dobbiamo ripetere il ciclo while per arrivare al risultato. Nell'esercizio 12.5 usiamo un contatore per contare quante volte eseguiamo il while, ma, a differenza del caso della sommatoria, ***non sappiamo in anticipo a quale valore arriverà***: in questo caso, è proprio il valore raggiunto dal contatore a darci il risultato.

In breve: anche se è utile osservare che esercizi si assomigliano tra loro, non esiste una regola generale per scrivere un ciclo while. Ricordatevi che ***ogni ciclo fa storia a sé***.

Inoltre, non vi spiegheremo perchè l'algoritmo per la radice quadrata funziona. In questo corso non vi chiederemo di trovare un algoritmo, tranne che nei casi più semplici. Un esercizio di solito consiste semplicemente nel ***trascrivere un algoritmo*** che vi diamo noi.

Esercizio 12.1. Scrivete una funzione

```
int SQ(int n)
```

che prende un intero n≥0, e restituisce la somma (1*1 + 2*2 + 3*3 + … + n*n) dei quadrati dei primi n interi. **Suggerimenti.** Usate una variabile intera s per "accumulare" i valori parziali della somma. **Esempi.** SQ(0)=0, SQ(1)=1, SQ(2)=5, SQ(3)=14, SQ(4)=30.

Esercizio 12.2. Scrivete una funzione

```
double Taylor(double x, int n)
```

che prende un reale x, un intero n≥0, e restituisce la somma:

$$\Sigma_{0\le i\le n}\ x^i/i! = x^0/0! + x^1/1! + x^2/2! + \dots + x^n/n!$$

Suggerimenti. Usate una variabile reale s per "accumulare" i valori parziali della somma. Usate le funzioni pow(x,y) e tgamma(x) della libreria math.h. Per calcolare il fattoriale usate la funzione ***"Gamma di Eulero"***, indicata in C++ con tgamma(x). Il fattoriale di un intero è uguale al valore di tgamma sull'intero dopo: per ogni intero v≥0, **v! = tgamma(v+1). Esempi.** Sia n=12. Taylor(1.,n) = 2,71828 (circa). Taylor(2.,n) = 7,38905 (circa).

Esercizio 12.3
Scrivete una funzione

double Taylor2(**double** x, **int** n)

che prende un reale x, un intero n≥0, e restituisce la somma:
$\Sigma_{0\le i\le n}(-1)^i x^{2i+1}/(2i+1)! = x^1/1! - x^3/3! + x^5/5! + \ldots + (-1)^n x^{2n+1}/(2n+1)!$
Suggerimenti. Usate i suggerimenti dell'esercizio precedente. **Esempi.** Sia pi=3.14159265, n=12. Allora Taylor2(pi/2,n) = +1 (circa). Taylor2(3*pi/2,n) = -1 (circa).

Esercizio 12.4. (Algoritmo "babilonese" per la radice quadrata). Siano x, epsilon due numeri reali > 0. Per "radice quadrata di x, a meno di epsilon" si intende un y tale che |y*y-x|<epsilon. Definite una funzione

double radice(**double** x, **double** epsilon)

che calcola la radice quadrata di x a meno di epsilon. **Suggerimento.** Partite da y=1 e ripetete l'operazione:

$y \mapsto (y + (x/y))/2$

finché trovate un y tale che |y*y-x|<epsilon. Per farlo, usate un ciclo **while** con condizione la negazione di (|y*y-a|<epsilon). La funzione "valore assoluto" si indica con **fabs(…). Esempio.** radice(2,0.0001)=1,4142 (circa), radice(3,0.0001)=1,7320 (circa). **Spiegazione.** Perché prendere la media tra y e x/y? Se y approssima la radice di x per eccesso, allora x/y la approssima per difetto, e prendere la media è un modo per *avvicinarsi al risultato*.

Esercizio 12.5. (Radice cubica con un metodo "ingenuo"). Definite una funzione

int cubic(**int** x)

che calcola la radice cubica di un intero arrotondata per difetto. **Suggerimento.** Partite da i=0 e cercate il primo valore di i per cui i^3>x. Trovato un tale i, la radice cubica arrotondata per difetto è i-1. Usate un ciclo **while** che controlla se $i^3 \le x$, e in caso affermativo incrementa i di 1. Quando i^3>x il **while** termina: ricordatevi di restituire i-1 come risultato. **Attenzione.** Evitare di usare pow(i,3) per calcolare i^3. Gli errori di approssimazione, che sono inevitabili per un calcolo su numeri reali, fanno si', per esempio, che pow(10,3) non sia esattamente 1000, ma solo molto vicino a 1000. Di conseguenza, a volte il test (pow(i,3)<=x) da' una risposta errata. Usate i*i*i (un valore intero esatto) al posto di pow(i,3). **Esempi.** cubic(1000)=10, cubic(1330)=10, cubic(1331)=11, cubic(1727)=11, cubic(1728)=12.

Lezione 12 - Esercizi su cicli While
Parte 2. Soluzioni

```
// Lezione12-Esempi di cicli while-Soluzioni
#include <math.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

/* 12.1. Somma dei primi n quadrati */
int SQ(int n) //Sia n>=0
{int i=1; int result=0;
 while (i<=n)
  {result=result+i*i;
   i=i+1;};
 return result;
}

/* 12.2. Sviluppo di Taylor 1 */
double Taylor(double x, int n) //Sia n>=0
{int i=0; double result=0;
 while (i<=n)
  {result=result+pow(x,i)/tgamma(i+1);
   i=i+1;};
 return result;
}

/* 12.3. Sviluppo di Taylor 2 */
double Taylor2(double x, int n) //Sia n>=0
{int i=0; double result=0;
 while (i<=n)
  {result=result+pow(-1.,i)*pow(x,2*i+1)/tgamma(2*i+2);
   i=i+1;};
 return result;
}

/* 12.4. Radice quadrata a meno di epsilon */
double radice(double x, double epsilon) //x>=0, epsilon > 0.
{double y=1.;
 while (fabs(y*y-x)>=epsilon)
  {y = (y+(x/y))/2;}
 return y;
}

/* 12.5. Radice cubica arrotondata per difetto */
int cubic(int x)
{int i=0;
 while (i*i*i<=x) {i=i+1;};
 return (i-1);
}
```

```
int main (){double x, epsilon;int n;

 cout << "TEST SQ(n)" << endl;
 n=0; cout << "SQ(" << n << ")=" << SQ(n) << endl;
 n=1; cout << "SQ(" << n << ")=" << SQ(n) << endl;
 n=2; cout << "SQ(" << n << ")=" << SQ(n) << endl;
 n=3; cout << "SQ(" << n << ")=" << SQ(n) << endl;
 system("PAUSE");system("CLS");

 cout << "TEST Taylor(x,n)" << endl;
 x=1.0;n=12; cout << "Taylor(" << x << ","
  << n << ")=" << Taylor(x,n) << endl;
 x=2.0;n=12; cout << "Taylor(" << x << ","
  << n << ")=" << Taylor(x,n) << endl;
 system("PAUSE");system("CLS");

 cout << "TEST Taylor2(x,n)" << endl; double pi=3.14159265;
 x=pi/2;  n=12; cout << "Taylor2(" << x << ","
  << n << ")=" << Taylor2(x,n) << endl;
 x=3*pi/2;n=12; cout << "Taylor2(" << x << ","
  << n << ")=" << Taylor2(x,n) << endl;
 system("PAUSE");system("CLS");

 cout << "TEST radice(x)" << endl;
 x=2.0; epsilon = 0.00001; cout << "radice(" << x << "," <<
epsilon << ")=" << radice(x,epsilon) << endl;
 x=3.0; epsilon = 0.00001; cout << "radice(" << x << "," <<
epsilon << ")=" << radice(x,epsilon) << endl;
 system("PAUSE");system("CLS");

cout << "TEST cubic(x)" << endl;
n=1000; cout << "cubic(" << n << ")=" << cubic(n) << endl;
n=1330; cout << "cubic(" << n << ")=" << cubic(n) << endl;
n=1331; cout << "cubic(" << n << ")=" << cubic(n) << endl;
n=1727; cout << "cubic(" << n << ")=" << cubic(n) << endl;
n=1728; cout << "cubic(" << n << ")=" << cubic(n) << endl;
system("PAUSE");system("CLS"); }
```

Tutorato 06 – esercitazione sui cicli while
Parte 1. Esercizi proposti

In questo tutorato vi chiediamo per tre volte di seguito di scrivere una funzione che decide se un numero è primo: ogni volta vi proponiamo una soluzione un poco più efficiente della precedente, anche se sempre molto lenta. Esistono degli ottimi algoritmi per decidere se un numero è primo, ma non li vedremo in questo corso.

Agli esercizi sui numeri primi aggiungiamo altri esercizi di argomento diverso, ma come al solito vi chiediamo di concentrarvi sui primi esercizi.

Tut06.1 Scrivete una funzione

bool PrimoUno(**int** n)

che restituisca **true** se e solo se n è un numero primo, come segue. Controllate se n è divisibile per qualche 2 <= d <= n-1. **Spiegazione.** Questa soluzione usa proprio la definizione di "essere primo". **Esempi.** Controllate che 2, 3, 5, 7, 37, 101 sono primi, mentre -1, 0, +1, 4, 6, 9, 25, 49 non lo sono.

Tut06.2 Scrivete una funzione

bool PrimoDue(**int** n)

che restituisca **true** se e solo se n è un numero primo, come segue. Controllate se n è divisibile per qualche 2 <= d <= sqrt(n). **Spiegazione.** Se n ha un divisore proprio d, allora n ha un divisore proprio d <= sqrt(n): questo semplice accorgimento risparmia molti calcoli. **Esempi.** Vedi sopra.

Tut06.3 Scrivete una funzione

bool PrimoTre(**int** n)

che restituisca **true** se e solo se n è un numero primo, come segue. Se n è pari, controllate se vale 2: questo è l'unico primo pari. Se invece n è dispari, controllate se n è divisibile per qualche dispari 3 <= d <= sqrt(n). Se lo è, n è composto, altrimenti n è primo. **Spiegazione.** Se n è dispari e ha un divisore proprio, allora n ha un divisore proprio d tale che (d dispari) e (d <= sqrt(n)). In questo modo risparmiamo altri calcoli. **Esempi.** Vedi sopra.

Tut06.4 Scrivete una funzione

void FattoriPrimi(**int** n)

che stampi i fattori primi di un intero n>0. **Suggerimento.** Un metodo semplice, anche se poco efficiente, è il seguente. Passate in rassegna tutti i numeri tra 2 ed n. Usate il test di primalita' dell'esercizio 3: ogni volta che trovate un fattore primo k stampatelo, poi dividete n per k. **Nota.** Non vi chiediamo di stampare più volte i fattori primi ripetuti, scegliete voi se volete farlo oppure no. **Esempi.** Controllate

le seguenti scomposizioni: 1 = (scomposizione vuota), 6=2*3, 7=7, 12=2*2*3, 16=2*2*2*2, 144=2*2*2*2*3*3.

Tut06.5 Scrivete una funzione

void MultGroup(**int** n)

che stampi i numeri naturali d tra 1 ed n coprimi con n. **Suggerimento.** Usate la funzione MCD(a,b) definita in una lezione precedente e controllate che sia MCD(d,n)=1. **Esempio.** Controllate che gli interi tra 1 e 10 e coprimi con 10 siano: 1, 3, 7, 9.

Tut06.6 Scrivete una funzione

void Radici(**int** n)

che stampi le radici n-esime complesse dell'unità, per n intero positivo. **Suggerimento.** Nel campo dei complessi per ogni intero positivo n esistono esattamente n radici n-esime dell'unità che sono della forma:

cos((2πk)/n)+ i*sen((2πk)/n)

per 0 <= k < n. Attenzione: *"sen(x)"* in inglese si scrive *"sin(x)"*. **Esempio.** Controllate che le 4 radici quarte dell'unita' sono, nell'ordine: +1, +i, -1, -i.

Tutorato 06
Parte 2. Soluzioni

Tut06.1 Scrivere una funzione

bool PrimoUno(**int** n)

che restituisca **true** se e solo se n è un numero primo, controllando se n è divisibile per qualche $2 \leq d \leq n-1$.

```
bool PrimoUno(int n){
  int k=2;
  while(k < n){
        if(n%k==0) return false;
        k=k+1;
  }
  return true;
}
```

Tut06.2 Scrivere una funzione

bool PrimoDue(**int** n)

che restituisca **true** se e solo se n è un numero primo, controllando se n è divisibile per qualche 2 <= d <= sqrt(n).

```
bool PrimoDue(int n){
  int k=2;
  while(k < sqrt(n)){
        if(n%k==0) return false;
        k=k+1;
  }
  return true;
}
```

Tut06.3 Scrivere una funzione

bool PrimoTre(**int** n)

che restituisca **true** se e solo se n è un numero primo, controllando se n è 2 oppure se è dispari ed è divisibile per qualche dispari 3 <= d <= sqrt(n).

```
bool PrimoTre(int n){
  if (n%2==0) return n==2;
  else{
  int k=3;
  while(k < sqrt(n)){
        if(n%k==0) return false;
        k=k+2;
  }
  return true;
}
}
```

Tut06.4 Scrivere una funzione

void FattoriPrimi(**int** n)

che stampi i fattori primi di un numero naturale n. La

soluzione inclusa stampa un fattore primo il numero di volte con cui compare, ma questo miglioramento non è richiesto.

```
void FattoriPrimi(int n){
     int k=2;
     int m=n;
     while(k <= m){
          if (PrimoTre(k) && m%k==0) {
          cout << k << ";";
          m = m/k;}
          else{ k++;}
          }
     cout << endl;
     }
```

Tut06.5 Scrivere una funzione

void MultGroup(**int** n)

che stampi i numeri naturali (minori di n) coprimi con n.

```
void MultGroup(int n){
  int k=1;
  while(k <= n){
        if (MCD(k,n) == 1) cout << k << ";";
        k++;
        }
  cout << endl;
  }
```

Tut06.6 Scrivere una funzione

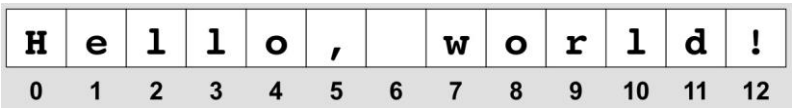
void Radici(**int** n)

che stampi le radici n-esime dell'unità. **Suggerimento**: nel campo dei complessi per ogni intero positivo n esistono esattamente n radici n-esime dell'unità che sono della forma: cos((2πk)/n)+ i*sen((2πk)/n), per 0 <= k < n.

```
const double Pi= acos(-1.0);

void Radici(int n){
     double k=0;
     double theta;
     while(k<n){
      theta = (2*k/n)*Pi;
      cout << "cos(" << 2*k/n << "Pi) + sin(" <<  2*k/n
      << "Pi) i" << "=" << cos(theta) << "+" << sin(theta)
      << "i" << endl;
      k=k+1;
     }
}
```

Settimana 07 – Stringhe
"Linee di caratteri"

1. **Lezione 13: Stringhe C++ (capitolo 7 libro testo)**
2. **Lezione 14: esercitazioni su stringhe**
3. **Tutorato 07: ricorsione e stringhe.**



"Hello, world!" è una stringa di 13 caratteri,
di solito la prima a essere presentatata in un corso di C++

Lezione 13 - Stringhe
Parte 1. Esempi (Cap.7 libro testo)

In questa seconda parte del corso tratteremo la rappresentazione dei dati di un programma attraverso tipi "composti", ovvero costruiti a partire da tipi più semplici. Il primo tipo composto che vediamo è il tipo **string** delle stringhe in C++. Le stringhe sono liste di caratteri ASCII (dunque di oggetti di tipo **char**) di lunghezza qualsiasi: a differenza dei tipi **int**, **bool**, **double**, non esiste una limitazione a priori alla loro dimensione, tranne la disponibilità di memoria del compilatore. *Insieme a una stringa dobbiamo sempre ricordarci di definire la sua lunghezza.*

Il C++ ha due tipi di stringhe: le stringhe C, costanti indicate tra doppie virgolette "abc", e le stringhe C++, indicate allo stesso modo, ma per cui sono disponibili molte funzioni predefinite: length, find, … . Le stringhe C++ costituiscono quello che viene chiamato in informatica una ***classe*** e richiedono una nuova libreria: <string>.

In questo corso dobbiamo solo sapere che le classi usano una notazione ***prefissa nel primo argomento***. Per esempio, abbiamo una funzione find per trovare la posizione del carattere c nella stringa s, ma anziché scrivere find(s,c) dobbiamo scrivere s.find(c). Consigliamo di ***assegnare il valore di s.find(c) a una variabile*** prima di usarlo, scrivendo per esempio:

```
int posizione_c = s.find(c);
```

Questo sia per leggibilità, sia perche' il Dev-C++ contiene purtroppo alcuni errori riguardo all'uso di espressioni che contengono notazioni come s.find(c). Quindi queste espressioni vanno evitate per quanto possibile.

Una variabile di tipo stringa si definisce con **string s = "abc";** dove abc sono i caratteri della stringa. Una variabile di tipo stringa con n caratteri uguali si definisce con **string s(n,'a');** dove 'a' è il carattere che viene ripetuto n volte in s.

La lunghezza di una stringa è il numero dei suoi caratteri: la stringa "" ha lunghezza 0. I caratteri di una stringa s sono numerati da 0 a (lunghezza di s)-1. Se s="abc", allora s[0] vale a, s[1] vale b, s[2] vale c. Ogni s[i] puo' essere usato come se fosse una variabile, ma dovete fare attenzione a non scrivere s[i] per i non compreso tra 0 e (lunghezza di s)-1. Cercare di utilizzare un s[i] inesistente produce errori nel programma.

Nel programma qui sotto i numeri da 7.1 a 7.13 fanno riferimento alle sezioni del capitolo 7 del libro di testo.

```
#include <math.h>
#include <iostream>
#include <stdlib.h>
// Dobbiamo includere una nuova libreria per le stringhe in C++
#include <string>

using namespace std;

// Nel capitolo 7.8 viene definita la funzione findFrom.
int findFrom (string s, char c, int i);
/* Noi non inseriamo subito la definizione di questa funzione. Per
ora, scrivendo una definizione con un punto e virgola al posto di
un {…}, ci limitiamo a "dichiarare" findFrom, cioè a descriverne i
tipi. Questo ci consente di usare findFrom nel main, ma è come
aver contratto un debito. Prima di eseguire il programma dovremo
inserire una definizione di findFrom da qualche parte, prima o
dopo il main. Altrimenti il compilatore ci segnalera' che findFrom
è indefinita. */


int main (){
/* 7.1. variabili di tipo stringa C++:
 contengono liste di caratteri ASCII di lunghezza qualsiasi */
cout << "Capitolo 7.1" << endl;
string first  = "Hello, ";
string second = "world.";
cout << first << second << endl;
//NOTA.Le stringhe costanti "abc" appartengono al tipo delle
//stringhe C ma vengono automaticamente convertite in stringhe C++
system("PAUSE");system("CLS");

/* 7.3. I caratteri di una stringa sono numerati da 0 a lungh.-1*/
cout << "Capitolo 7.3" << endl;
string fruit = "banana";
char letter0 = fruit[0];
char letter1 = fruit[1];
cout << fruit << endl;
cout << "letter0 = "  << letter0 << endl;
cout << "letter1 = "  << letter1 << endl;
system("PAUSE");system("CLS");

/* 7.4. L'operazione (.).length()
  Calcola il numero di caratteri di una stringa */
cout << "Capitolo 7.4" << endl;
int length;
length  =  fruit.length();  /*  il  primo  argomento  di  length  è
prefisso cioe' è scritto prima di length. "()" indica gli altri
argomenti di "length", in questo caso nessuno. */
cout << "length(" << fruit << ")=" << length << endl;
char last = fruit[length-1];
cout << "last = " << last << endl;
```

```
/* stampare un carattere di indice non in [0,length-1] produce un
errore oppure un valore casuale */
//cout << fruit[-1];
//cout << fruit[length];
system("PAUSE");system("CLS");

/* 7.5. Un while che "attraversa" una stringa */
cout << "Capitolo 7.5" << endl;
cout << "stampa di:\n\t" << fruit << endl <<
 "carattere per carattere e in verticale" << endl;
int index = 0;
while (index < fruit.length())
 {
  char letter = fruit[index];
  cout << letter << endl;
  index = index + 1;
 }
system("PAUSE");system("CLS");

/* 7.7 La funzione find
 La funzione (.).find(.) trova il primo indice in cui compare una
lettera (scritta tra apici: 'a') o stringa (scritta tra doppi
apici: "aaa"). Se la lettera o stringa non c'e' restituisce -1 */
cout << "Capitolo 7.7" << endl;
int index_first_a = fruit.find('a'); //il primo argomento di find
//è prefisso, cioe' scritto prima, della funzione find.
cout << "index_first_a =   " << index_first_a << endl;

int index_first_z = fruit.find('z');
cout << "index_first_z =   " << index_first_z << endl;

int index_first_nan = fruit.find("nan");
cout << "index_first_nan = " << index_first_nan << endl;
system("PAUSE");system("CLS");

/* 7.8. Una nostra versione della funzione find */
cout << "Capitolo 7.8" << endl;
cout << "Cerco il primo indice del carattere a in banana"
 << endl << "a partire dall'indice 4:" << endl;
cout << "  findFrom(banana,a,4)  =  " << findFrom(fruit,'a',4) <<
endl;
system("PAUSE");system("CLS");

/* 7.9 Conto di una lettera in una stringa
  Il valore della stringa fruit è "banana" */
cout << "Capitolo 7.9" << endl;
cout << "Numero di lettere a in banana: ";
int i = 0; int count = 0;
while (i < fruit.length())
 {
  if (fruit[i] == 'a')
   {count = count + 1;}
```

```
  i = i + 1;
 }
cout << count << endl;
system("PAUSE");system("CLS");

/* 7.11. Concatenazione di stringhe: l'operazione + */
cout << "Capitolo 7.13" << endl;
string bakedGood = " nut bread";
string dessert = fruit + bakedGood;
cout << "dessert = " << dessert << endl;
/* viene stampato "fruit nut bread": le due stringhe sono state
concatenate */
system("PAUSE");system("CLS");

//non possiamo invece applicare + a due stringhe costanti:
// cout << "banana" + " nut bread";
//Il motivo è che le costanti appartengono a un vecchio
//tipo di stringhe, le stringhe C, su cui il + non è definito

/*7.12. Modifica di stringhe */
cout << "Capitolo 7.12" << endl;
string greeting = "Hello, world!";
cout << "greeting = " << greeting << endl;
greeting[0] = 'J';
cout << "greeting = " << greeting << endl;
// viene stampato "Jello, world"
system("PAUSE");system("CLS");

/*7.13. Confronto di stringhe: l'ordine è l'ordine lessicografico,
quello tra le parole nel  vocabolario */
cout << "Capitolo 7.13" << endl;
string s1 = "abc"; string s2 = "b"; //si confronta a con b
if (s1 < s2)
 {cout << s1 << " viene prima di " << s2 << endl;}
else
 {cout << s2 << " viene prima di " << s1 << endl;}
// viene stampato "abc viene prima di b"

s1 = "abc"; s2 = " abc"; //si confronta a con il carattere vuoto
if (s1 < s2)
 {cout << s1 << " viene prima di " << s2 << endl;}
else
 {cout << s2 << " viene prima di " << s1 << endl;}
// viene stampato " abc viene prima di abb"
system("PAUSE");system("CLS");
}

/* Capitolo 7.8: una definizione di findFrom(s,c,i), con uscita
prematura da un ciclo while.
```

Inseriamo la definizione promessa di find all'inizio del main. Cerco il primo indice del carattere c nella stringa s a partire dall'indice i (e non dall'indice 0).
Usiamo un return all'interno del ciclo while per consentire una uscita prematura dal ciclo non appena troviamo il carattere c in s */

```
int findFrom (string s, char c, int i)
{ /* Notate che la variabile i riceve un valore attraverso la chiamata di funzione: non deve essere assegnata, altrimenti cancelliamo il valore che i riceve dal main */
  while (i<s.length())
   {if (s[i] == c)
     return i;  //se trovo c esco subito dal while, restituendo i
    i = i + 1;} //incremento
  return -1;}   //nel caso non trovo c alla fine restituisco -1
```

Lezione 13 - Stringhe
Parte 2. Esercizi

Vi raccomandiamo di concentrarvi sui primi tre esercizi. Tutti gli esercizi che seguono si risolvono adattando il ciclo di conto del carattere 'a':

```
int i = 0; int count = 0;
while (i < s.length()) /* occhio a non dimenticare il "()" */
 {
  if (s[i] == 'a')
   {count = count + 1;}
  i = i + 1;
 }
```

appena visto a lezione. Come esempio, in tutti gli esercizi che seguono consideriamo la bizzarra stringa seguente:

string s = "abc 123 \n @#$ \t abc567ABC"

Alla fine della lezione vedremo una ***soluzione più efficiente per l'esercizio 13.2*** (ricerca di un carattere in una stringa), che utilizza l'uscita prematura da un ciclo.

13.1. Conto di quante volte un carattere compare in una stringa. Scrivete una funzione

int conta(**char** c, **string** s)

che dato un carattere c e una stringa s conta quante volte c appartiene ad s. **Esempi *(con s definita qui sopra):*** conta('a',s) = 2, conta('z',s) = 0, conta('#',s) = 1.

13.2. Appartenenza di un carattere a una stringa. Scrivete una funzione

bool appartiene(**char** c, **string** s)

che dato un carattere c e una stringa s restituisce vero se c appartiene ad s e falso altrimenti. **Suggerimento.** Usate una variabile booleana result, che parte da **false**, e che viene aggiornata a **true** quando trovate un s[i] uguale a c. Usciti dal while result contiene il risultato. **Esempi.** appartiene('a',s)=1, appartiene('z',s)=0, appartiene('#',s)=1.

13.3. Conto degli spazi bianchi una stringa. Chiamiamo spazio un carattere della forma ***' ' (spazio bianco)*** oppure ***'\n' (newline)*** oppure ***'\t' (tabulazione)***. Scrivete una funzione

int contaspazi(**string** s)

che data una stringa s conta quanti spazi compaiono in s. **Nota.** Una tabulazione '\t' conta 1, non importa quanti spazi bianchi produca sullo schermo. **Esempio.** contaspazi(s)=7.

13.4. Conto delle cifre in una stringa. Una cifra è uno dei 10 caratteri della forma: ***'0', …, '9'***. Una definizione alternativa utile è la seguente: una cifra è un carattere c tale che ***('0'≤c) e (c≤'9')*** nell'ordine dell'alfabeto ASCII. Scrivete una funzione

int contacifre(**string** s)

che data una stringa s conta quante cifre compaiono in s. **Esempio.** contacifre(s)=6.

13.5. Conto delle minuscole in una stringa. Una minuscola è uno dei 26 caratteri dell'alfabeto inglese: ***'a', …, 'z'***. Una definizione alternativa utile è la seguente: una minuscola è un carattere c tale che ***('a'≤c) e (c≤'z')*** nell'ordine dell'alfabeto ASCII. Scrivete una funzione

int contaminuscole(**string** s)

che data una stringa s conta quante cifre compaiono in s. **Esempio.** contaminuscole(s)=6.

13.6. Conto delle parole *(questo esercizio è più impegnativo e non è richiesto)*. Definiamo "spazi" i caratteri seguenti: ***' ', '\n', '\t'***. Definiamo ***"parola"*** ogni gruppo di caratteri che ***non*** sono spazi, che sia preceduto da uno spazio oppure dall'inizio della stringa, e che sia seguito da uno spazio oppure dalla fine della stringa. Scrivete una funzione

int contaparole(**string** s)

che data una stringa s conta quante parole (secondo la definizione data di parole) compaiono in s.

Suggerimento. Le parole sono in corrispondenza uno-a-uno con i loro ultimi caratteri. Quindi è sufficiente contare i caratteri di s che si trovano alla fine di una parola. Per contare questi ultimi, contate i caratteri che soddisfano ***entrambe*** queste condizioni:

1. ***non*** sono spazi
2. sono l'ultimo carattere di s, oppure, se non lo sono, allora il carattere successivo in s è uno spazio.

Per spiegare queste due condizioni basta guardare alcuni esempi.

Esempi. Negli esempi seguenti indichiamo in rosso i caratteri alla fine di una parola: sono tanti quanti le parole. contaparole(" ")=0, contaparole(" ab**c**")=1, contaparole(" ab**c** de**f** ")=2, contaparole(" ab**c** de**f** x**y**")=3. Se s = "ab**c** 12**3** \n @#**$** \t abc567AB**C**", allora contaparole(s)=4. Notate che il salto di riga isolato "\n" è uno spazio, quindi non conta come parola.

Lezione 13 - Stringhe
Parte 3. Soluzioni

```
// Lezione13-Stringhe C++-Soluzioni
#include <math.h>
#include <iostream>
#include <stdlib.h>
#include <string>
using namespace std;

/* 13.1. Conto di un carattere */
int conta(char c, string s)
{int i=0; int count=0;
 int lun = s.length();  /* occhio a non dimenticare il "()" */
 while (i<lun)
  {if (s[i]==c)
    {count = count + 1;}
   i=i+1;}
 return count;
};

/* 13.2. Appartenenza di un carattere */
bool appartiene(char c, string s)
{int i=0; bool result=false;
 int lun = s.length();  /* occhio a non dimenticare il "()" */
 while (i<lun)
  {if (s[i]==c)
    {result=true; /*mi annoto che ho trovato c */ }
   i=i+1;}
/*se non ho trovato c allora return vale ancora false*/
 return result; /*se invece ho trovato c, result vale true */
};

/* 13.3. Conto degli spazi */
/* definisco un test che controlla se un carattere è uno spazio.*/
bool spazio(char c)
{return ((c == ' ') || (c == '\n') || (c == '\t'));}

int contaspazi(string s)
{int i=0; int count=0;
 int lun = s.length(); /* occhio a non dimenticare il "()" */
 while (i<lun)
  {if (spazio(s[i])==true)
    {count = count + 1;}
   i=i+1;}
 return count;
};


/* 13.4. Conto delle cifre */
/* definisco un test che controlla se un carattere è una cifra */
bool cifra(char c)
```

```
{return ((c >= '0') && (c <= '9'));}

int contacifre(string s)
{int i=0; int count=0; int lun = s.length();
 while (i<lun)
  {if (cifra(s[i])==true)
    {count = count + 1;}
   i=i+1;}
 return count;
};


/* 13.5. Conto delle minuscole
Definisco un test che controlla se un carattere è una minuscola */
bool minuscola(char c)
{return ((c >= 'a') && (c <= 'z'));}

int contaminuscole(string s)
{int i=0; int count=0; int lun = s.length();
 while (i<lun)
  {if (minuscola(s[i])==true)
    {count = count + 1;}
   i=i+1;}
 return count;
};

/* 13.6. Conto delle parole*/
int contaparole(string s)
{int i=0; int count=0; int lun = s.length();
 while (i<lun)
  {bool testfineparola =
        (spazio(s[i])==false) &&
        ((i==(lun-1)) || (spazio(s[i+1])==true));
   if (testfineparola == true)
    {count = count + 1;}
   i=i+1;}
 return count;
};

int main(){
string s = "abc 123 \n @#$ \t abc567ABC";
char c;

 cout << "Prova conta(c,s)" << endl;
 c='a'; cout << " s = " << s << endl << " c = " << c << endl;
 cout << " conta(c,s) = " << conta(c,s) << endl << endl;
 c='z'; cout << " s = " << s << endl << " c = " << c << endl;
 cout << " conta(c,s) = " << conta(c,s) << endl << endl;
 c='#'; cout << " s = " << s << endl << " c = " << c << endl;
 cout << " conta(c,s) = " << conta(c,s) << endl << endl;
 system("pause");system("CLS");
```

```
 cout << "Prova appartiene(c,s)" << endl;
 c='a'; cout << " s = " << s << endl << " c = " << c << endl;
 cout << " appartiene(c,s) = " << appartiene(c,s) << endl << endl;
 c='z'; cout << " s = " << s << endl << " c = " << c << endl;
 cout << " appartiene(c,s) = " << appartiene(c,s) << endl << endl;
 c='#'; cout << " s = " << s << endl << " c = " << c << endl;
 cout << " appartiene(c,s) = " << appartiene(c,s) << endl << endl;
 system("pause");system("CLS");

 cout << "Prova contaspazi(s)" << endl;
 c=' ';  cout << " c = " << c << " spazio(c) = " << spazio(c) <<
endl;
 c='\n'; cout << " c = " << c << " spazio(c) = " << spazio(c) <<
endl;
 c='\t'; cout << " c = " << c << " spazio(c) = " << spazio(c) <<
endl;
 c='a';  cout << " c = " << c << " spazio(c) = " << spazio(c) <<
endl;
 c='1';  cout << " c = " << c << " spazio(c) = " << spazio(c) <<
endl;
 cout << endl << " s = " << s << endl;
 cout << " contaspazi(s) = " << contaspazi(s) << endl << endl;
 system("pause");system("CLS");

 cout << "Prova contacifre(s)" << endl;
 c='1';  cout << " c = " << c << " cifra(c) = " << cifra(c) <<
endl;
 c='a';  cout << " c = " << c << " cifra(c) = " << cifra(c) <<
endl;
 cout << endl << " s = " << s << endl;
 cout << " contacifre(s) = " << contacifre(s) << endl << endl;
 system("pause");system("CLS");

 cout << "Prova contaminuscole(s)" << endl;
 c='a';   cout << " c = " << c << " minuscola(c) = " <<
minuscola(c) << endl;
 c='1';   cout << " c = " << c << " minuscola(c) = " <<
minuscola(c) << endl;
 c='A';   cout << " c = " << c << " minuscola(c) = " <<
minuscola(c) << endl;
 cout << endl << " s = " << s << endl;
 cout << " contaminuscole(s) = " << contaminuscole(s) << endl <<
endl;
 system("pause");system("CLS");

 cout << "Prova contaparole(s)" << endl;
 cout << " s = " << s << endl;
 cout << " contaparole(s) = " << contaparole(s) << endl << endl;
 system("pause");system("CLS");
}
```

L'uscita prematura da un ciclo: una soluzione efficente per l'esercizio 13.2.

Nell'esercizio 13.2 dobbiamo definire una funzione che restituisce **vero** se una stringa contiene un elemento uguale a un certo carattere c, e **falso** altrimenti. Si tratta di un problema comune: definire una funzione che restituisce ***vero se un certo insieme finito contiene un elemento con una certa proprietà***, e **falso** nel caso opposto, quando non ce ne sono.

Una soluzione efficiente, che evita di esaminare sempre tutta la stringa, è la seguente. Si utilizza un ciclo per passare in rassegna tutti gli elementi dell'insieme. Per ognuno di essi si controlla se l'elemento soddisfa la proprietà richiesta. ***Non appena*** si trova un elemento che soddisfa la proprietà, si termina subito il calcolo della funzione con un **return true**: in questo modo si evita di esaminare il resto della stringa. Se il ciclo finisce senza che si sia restituito vero, allora nessun elemento dell'insieme soddisfa la proprietà. In questo caso si termina il calcolo della funzione con un **return false**. Ecco la soluzione.

```
/* 13.2. Appartenenza di un carattere: versione più efficente */
bool appartiene(char c, string s)
{int i=0; int lun = s.length();
 while (i<lun)
  {if (s[i]==c)
    {return true; /*non appena trovo c esco con true*/ }
   i=i+1;}
/*se i arriva a lun e non ho trovato c esco con false*/
 return false;};
```

Se usate questa soluzione, ***un errore da evitare***, ma che capita spesso, è inserire all'interno del ciclo sia il **return true** che il **return false**. Supponiamo di risolvere l'esercizio 13.2 così:

```
bool appartiene(char c, string s) /* ← VERSIONE ERRATA !!! */
{int i=0; int lun = s.length();
 while (i<lun)
  {if (s[i]==c)
    {return true;}
   else
    {return false;} /* ← QUESTO RETURN È ERRATO !!! */
   i=i+1;}};
```

In questa soluzione (errata) quando i=0 usciamo comunque dal ciclo, sia che s[0]=c sia vero, sia che s[0]=c sia falso. Nel secondo caso sosteniamo che appartiene(c,s)=falso, ma non è giusto. Infatti anche se s[0]=c è falso, potrebbe ancora essere s[1]=c vero, o s[2]=c vero, eccetera. Quindi non sappiamo ancora se appartiene(c,s)=falso, e non possiamo inserire un **return false** all'interno del ciclo. Solo dopo aver terminato il ciclo sappiamo che s[i]=c è falso per ogni i, e possiamo scrivere **return false**.

Lezione 14 - Stringhe
Parte 1. Esercizi

Tutti gli esercizi che seguono hanno difficoltà paragonabile a un esercizio di esame, e tutti si risolvono adattando i cicli su stringhe degli esercizi della Lezione 13.

Nota 1. In questa lezione utilizziamo **i++** come abbreviazione di **i=i+1**. Usiamo questa abbreviazione solo perche' è di uso comune e può capitare di leggere un programma che la contiene.

Nota 2. Nell'esercizio 14.5, definiamo variabile di tipo stringa con n caratteri uguali allo spazio vuoto con: **string s(n,' ');**

14.1. Trasformazione in maiuscolo dei caratteri alfabetici di una stringa. Scrivete una funzione

string stringToUpper(**string** s)

che data una stringa s ne trasformi in maiuscolo tutti i caratteri alfabetici minuscoli. **Suggerimento.** Il carattere maiuscolo corrispondente al valore del parametro c è calcolato dalla funzione **char** toupper(**char** c). Se c non è una minuscola allora toupper(c) = c. **Esempi.** stringToUpper("Saltapicchio #91") = "SALTAPICCHIO #91".

14.2. Prefisso di una stringa. Date le stringhe s e t si dice che s è un prefisso di t se i caratteri iniziali di t formano s. Per esempio, s = "ab" è un prefisso di t = "abcd":

```
s = "ab"
     ||
t = "abcd" (cerco i caratteri di s in t)
```

La definizione rigorosa è: s è un prefisso di t se per ogni j tra 0 ed s.length() - 1, vale la congiunzione delle condizioni: j è un indice di s (cioe' j<t.length()), e (s[j] == t[j]).

Scrivete una funzione

bool prefix(**string** s, **string** t)

che decida se s sia prefisso di t. **Esempi.** prefix("abc","**abc**d") = **true**, mentre prefix("bcd","a**bcd**") = **false**, perche' "bcd" si trova sì in "abcd", ma non a partire dalla prima posizione. Infine prefix("abcd", "abc") = **false**, perche' la stringa "abcd" è piu' lunga della stringa "abc" e dunque non puo' trovarsi in essa.

14.3. Prefisso a partire da indice dato. Date le stringhe s e t ed un indice i compreso tra 0 e t.length() - 1 si dice che s è un prefisso di t da i in poi se i caratteri di t a partire dalla posizione i formano s. Per esempio, s = "bc" è un prefisso di t = "abcd" a partire da **i=1**:

s = "**bc**"
||
t = "a**bc**d" *(cerco i caratteri di s in t dalla posizione i=1)*

La definizione rigorosa è: s è un prefisso di t se per ogni j tra 0 ed s.length() - 1 vale la congiunzione delle condizioni: i+j è un indice di t (cioe' i+j<t.length()) e (s[j] == t[i+j]).

Scrivete una funzione

bool prefix(**string** s, **string** t, **int** i)

che decida se s sia prefisso di t da i in poi. **Esempi.** prefix("tisca", "ba**tisca**fo",2) = **true**. Invece prefix("abc", "**abc**d",1) = **false**, perche' "abc" si trova sì in "abcd", ma a partire dalla posizione i=0, non da i=1. Infine prefix("cde", "abcd",2) = **false**, perche' a partire da i=2 la stringa "abcd" ha solo due caratteri e quindi non puo' contenere "cde".

14.4 Sottostringa. Date due stringhe s e t si dice che s è una sottostringa di t se s è contenuta (consecutivamente) in t, ovvero se esistono le stringhe u e v tali che t = u + s + v. Scrivete una funzione

int substring(**string** s, **string** t)

che restituisce il minimo indice a partire dal quale s è sottostringa di t, se esiste, -1 altrimenti. **Suggerimento.** Usate la funzione prefix(s,t,i) per stabile se s è sottostringa di t a partire da un indice i, e un ciclo **while** per provare tutti i valori di i che sono indici di t. Appena trovate un tale i terminate il calcolo e restituite i. Se arrivate alla fine del **while**, e dunque non avete trovato un tale i, restituite invece -1.

14.5 Inversione di una stringa*(questo esercizio, più impegnativo, non è richiesto)*. Una stringa è l'inversa di s se consiste degli stessi caratteri di s, ma in ordine inverso. Scrivete la funzione

string reverse(**string** s)

che ritorna l'inversa di s. **Esempio.** reverse("abcde")="edcba". **Suggerimento.** Potete usare uno degli algoritmi seguenti.

Algoritmo 1. Calcolate la lunghezza l=s.lenght(); di s. Potete definire una variabile t di tipo stringa e assegnarvi una stringa di spazi bianchi della stessa lunghezza l di s con il comando: **string** t(l,' '). Copiate i caratteri di s su t ponendo il carattere di posto 0 nel posto l-1, quello di posto 1 nel posto l-2, e così via. Alla fine restituite t.

Algoritmo 2. Senza creare alcuna stringa diversa da s, definite due indici i e j inizializzati rispettivamente a 0 e a s.length()-1; fintanto che (i<j) scambiate s[i] con s[j] *(non continuate quando i≥j, altrimenti rimettete i caratteri al loro posto!)*. Alla fine restituite s.

Lezione 14 - Stringhe
Parte 2. Soluzioni

Una osservazione sulle soluzioni di 14.2 e 14.3. Prestate particolare attenzione alle soluzioni degli esercizi 14.2, 14.3. In essi dobbiamo stabilire se per tutti i valori di j tra 0 e t.length()-1 vale una certa proprietà. Si tratta di un problema simile, ma non identico, a quello visto nell'esercizio 13.2, quando si trattava di stabilire se almeno uno degli elementi di un insieme soddisfaceva una certa proprietà.

La soluzione di 14.2, 14.3 è simile a quella già vista per 13.2. Scorriamo con un ciclo tutti i valori di j, e per ognuno di essi controlliamo se la proprietà vale. Se la proprietà non vale anche solo una volta terminiamo il calcolo della funzione con un **return false:** l'uscita prematura si giustifica con il fatto che sappiamo con certezza che il risultato finale è **false**. Se invece il ciclo termina senza che sia stato usato il **return false;** allora tutti i valori di j soddisfano la proprietà richiesta. In questo caso terminiamo il calcolo della funzione con un **return true:** ma non scritto dentro il ciclo while, subito dopo.

Infatti se inserite il **return true** dentro il ciclo while, potete uscire dal ciclo prima di sapere con certezza che la proprietà vale per tutti gli indici j, e fornire un risultato errato.

```
/* Lezione14-Stringhe.
   Soluzioni esercizi sulle stringhe-Soluzioni */

#include <iostream>
#include <stdlib.h>
using namespace std;

/* 14.1. Trasformazione in maiuscolo */
string stringToUpper(string s)
{
      int index = 0;
      while (index < s.length())
      {
            s[index] = toupper(s[index]);
            index++;
      }
      return s;
}

/*14.2. Prefisso*/
bool prefix(string s, string t)
// post: true se s è un prefisso della stringa t
{
 int len_s = s.length(), len_t = t.length();
```

```
 int j = 0; // j indice su s
 while (j < len_s)
  { /* se anche per un solo valore di j la coppia di condizioni
relativa a j è falsa esco dal ciclo e restituisco "false" */
   bool prefix_conditions = (j < len_t) && (s[j] == t[j]);
   if (prefix_conditions == false)
    return false;
   j++;
  }
/* se il ciclo termina, allora le condizioni per tutti i valori di
j sono vere, quindi restituisco "true" */
 return true;
}


/* 14.3. Prefisso da un indice i in poi */
bool prefix(string s, string t, int i)
/* Supponiamo i>=0. Ottengo true se s è un prefisso della stringa
t[i..t.length()-1], e false altrimenti */
{
 int len_s = s.length(), len_t = t.length();
 int j = 0; // j indice su s, mentre (i+j) è un indice su t
 while (j < len_s)
  { /* se anche per un solo valore di j la coppia di condizioni
relativa a j è falsa esco dal ciclo e restituisco "false" */
   bool prefix_conditions = (i+j < len_t) && (s[j] == t[i+j]);
   if (prefix_conditions == false)
    return false;
   j++;
  }
/* se il ciclo termina, allora le condizioni per tutti i valori di
j sono vere, quindi restituisco "true" */
 return true;
}

/*14.4. Sottostringa*/
int substring(string s, string t)
// post: indice da cui s è contenuta in t (prima occ.) se esiste;
//       -1 altrimenti
{     int i = 0;
      while (i < t.length())
       {if (prefix(s, t, i)==true)
                return i;
        i++;}
      return -1;
}

/*14.5. Inversione */
string strReverse1(string s)
// post: ritorna l'inversa di s
{
    int len_s = s.length();
```

```
    string t(len_s, ' ');
// crea una stringa della lunghezza richiesta, tutta fatta di ' '
    int i = 0;
    while (i < len_s)
    {
          t[len_s-i-1] = s[i];
          i++;
    }
    return t;
}

string strReverse2(string s)
// post: ritorna l'inversa di s
{
     int i = 0, j = s.length() - 1;
     while (i < j)
     {    char c = s[i]; // scambia s[i] con s[j]
          s[i] = s[j];
          s[j] = c;
          i++;
          j--;
     }
     return s;
}

int main()
{cout << "14.1. TEST stringToUpper" << endl;
 string str = "Pippo 9"; string str0 = "Saltapicchio #91";
 cout<<" stringToUpper("<<str<<")="<<stringToUpper(str)<<endl;
 cout<<" stringToUpper("<<str0<<")="<<stringToUpper(str0)<<endl;
 system("pause");system("CLS");

 cout << "14.2. TEST prefix " << endl;
 string str1 = "abcd";
 string str2 = "abc";
 string str3 = "bcd";
 cout << " prefix(" << str2 << "," << str1 << ")="
     << prefix(str2, str1) << endl;
 cout << " prefix(" << str3 << "," << str1 << ")="
     << prefix(str3, str1) << endl;
 cout << " prefix(" << str1 << "," << str2 << ")="
     << prefix(str1, str2) << endl;
 system("pause"); system("CLS");

 cout << "14.3. TEST prefix " << endl;
 string strA = "batiscafo";
 string strB = "bat";
 string strC = "tisca";
 cout << " prefix(" << strB << "," << strA << ", 0)="
     << prefix(strB, strA, 0) << endl;
 cout << " prefix(" << strB << "," << strA << ", 1)="
     << prefix(strB, strA, 1) << endl;
```

```
cout << " prefix(" << strC << "," << strA << ", 1)="
    << prefix(strC, strA, 1) << endl;
cout << " prefix(" << strC << "," << strA << ", 2)="
    << prefix(strC, strA, 2) << endl;
system("pause"); system("CLS");

cout << "14.4. TEST substring " << endl;
cout << " substring(" << strC << "," << strA << ")="
    << substring(strC, strA) << endl;
cout << " substring(" << str1 << "," << strA << ")="
    << substring(str1, strA) << endl;
system("pause"); system("CLS");

cout << "14.5 TEST strReverse1, strReverse2" << endl;
string s = "ambarabaciccicocco";
cout << " strReverse1(" << s << ")=" << strReverse1(s) << endl;
cout << " strReverse2(" << s << ")=" << strReverse2(s) << endl;
system("pause");system("CLS");}
```

Tutorato 07 – ricorsione e stringhe
Parte 1. Esercizi proposti

Uno degli esercizi, il secondo, richiede di definire un funzione ricorsiva: dovete trascrivere in C++ la definizione induttiva data come suggerimento.

Tut07.1. Scrivete una funzione

void minmax(){…};

che se chiamata richiede una serie di numeri reali da tastiera, e a ogni passo stampa il minimo e il massimo di tutti i numeri finora inseriti. La serie si interrompe quando viene inserito il numero 0. **Suggerimento.** Usate due variabili reali m, M per contenere il minimo e il massimo. Fate leggere il primo numero x, e assegnate x ad m, M e stampate m, M. Usate un ciclo **while** che continua finche' x è diverso da 0, e ogni volta fate leggere di nuovo x e assegnate a m il valore min(m,x), ad M il valore max(M,x) e e stampate m, M. **Esempio.** Provate la funzione con una lunga serie di numeri a caso.

Tut07.2. Scrivere una funzione ricorsiva che calcoli la somma di due interi n, m>=0, senza usare la somma predefinita, ma usando solo l'operazione di successore. **Suggerimento.** Usate la seguente definizione induttiva: Somma(n,m) = n se m=0, Somma(n,m) = Somma(n,m-1) + 1 altrimenti. Esempi. Controllate che Somma(0,2) = Somma(2,0) = 2, Somma(1,2) = Somma(2,1) = 3, Somma(2,2) = 4.

Tut07.3. Stringhe palindrome. Una stringa s è ***palindroma*** se i caratteri di s letti da sinistra a destra o da destra a sinistra producono lo stesso testo: è il caso di "osso", che letto al contrario resta "osso", ma non di "orso", che letto al contrario diventa "osro". Scrivete una funzione

bool pal(string s)

che prenda una stringa s e restituisca "**true**" se il testo s è palindroma, e "**false**" altrimenti. **Suggerimento.** Utilizzate un ciclo **while** e due indici i, j che partono rispettivamente da 0 e dalla lunghezza di s meno 1, e che indicano sempre due caratteri di posizione opposta rispetto al centro della stringa. A ogni passo controllate se s[i] è uguale a s[j]. Se i due caratteri sono diversi restituite subito **false** come valore della funzione, altrimenti incrementate i di 1 e decrementate j di 1. Continuate il **while** finche' (i<=j), e se arrivate alla fine del **while** (cioe' se non vi capita di restituire **false**) allora restituite **true**. **Esempi.** Controllate che sono palindromi*: "**osso**", "**radar**", "**kajak**" "**ailatiditalia**", "**itopinonavevanonipoti**"* e la lunghissima frase:

*"**omordotuanuoraoarounautodromo**"*

scritte senza spazi vuoti. Controllate che non sono palindromi: ***"orso", "asso", "alina"***.

Tut07.4. Date due stringhe, scrivete una funzione

bool contenuta(**string** s1, **string** s2)

che controlla se s1 è contenuta in s2, anche in modo non consecutivo ma nello stesso ordine. Per esempio: "abc" è contenuta in "a1b2c3" in modo anche non consecutivo ma nello stesso ordine. **Suggerimento.** Usate un indice i1 che percorre s1 e un indice i2 che percorre s2: entrambi partono da 0. Finche' i1 è < della lunghezza di s1 e i2 è < della lunghezza di s2, continuate a confrontare s1[i1] con s2[i2], se sono diversi incrementate i2 di 1, se sono uguali incrementate i1, i2 di 1. Quando il ciclo termina, se (i1 >= lunghezza di s1) allora abbiamo trovato ogni lettera di s1 in s2, quindi s1 è contenuta in s2: restituite **true.** Altrimenti abbiamo necessariamente: i1 < lunghezza s1, e quindi, dato che il ciclo è terminato, i2 >= lunghezza s2. In questo caso non abbiamo trovato il carattere numero i1 di s1 in s2: restituite **false. Esempi.** s1=NERO, s2=GANTTEORO, contenuta(s1,s2) = **true.** s1=NERO, s2=GANTTEOO, contenuta(s1,s2) = **false.**

Tutorato 07
Parte 2. Soluzioni

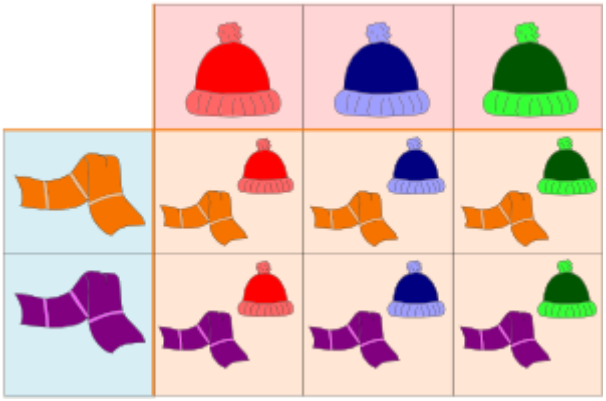
```
#include <iostream>
#include <stdlib.h>
using namespace std;

/* Tut07.1. Minimo e massimo di una sequenza di numeri reali */
void minmax()
{double x, m, M;
 cout << "Inserite una serie di numeri reali, terminata da 0" <<
endl;

 cin >> x; m = x; M = x;
 cout << " min = " << m << " max = " << M << endl;

 while (x!=0)
 {
  cin >> x;m = min(m,x); M = max(M,x);
  cout << " min = " << m << " max = " << M << endl;
 }
}

/* Tut07.2. Somma ricorsiva */
int Somma(int n, int m)
{
 if (m == 0)
  {return n;}
 else
  {return Somma(n,m-1)+1;}
}

/* Tut07.3. Stringhe palindrome */
bool pal(string s)
{int i=0, l = s.length(); int j = l-1;
 while ( (i<=j) )
  {
   if (s[i]!=s[j])
    {return false;}
   else
    {i=i+1; j=j-1;}
  }
 return true;
}

/* Tut07.4. Stringhe contenute */
bool contenuta(string s1, string s2)
{int i1=0, i2=0, l1 = s1.length(), l2 = s2.length();
 while ( (i1<l1) && (i2<l2) )
  {
   if (s1[i1]==s2[i2])
```

```
    {i1=i1+1; i2=i2+1;}
   else
    {i2=i2+1;}
  }
if (i1 >= l1)
 {return true;}
else
 {return false;}
}


int main(){string s, s1, s2; int n,m;
 cout << "Tutorato 07" << endl;

 cout << "Prova minmax" << endl;
 minmax();
 system("pause");system("CLS");

 cout << "Prova Somma(n,m)" << endl;
 n=0; m=2;
 cout << " n = " << n << " m = " << m << " Somma(n,m) = " <<
  Somma(n,m) << endl;
 n=2; m=0;
 cout << " n = " << n << " m = " << m << " Somma(n,m) = " <<
  Somma(n,m) << endl;
 n=2; m=1;
 cout << " n = " << n << " m = " << m << " Somma(n,m) = " <<
  Somma(n,m) << endl;
 n=1; m=2;
 cout << " n = " << n << " m = " << m << " Somma(n,m) = " <<
  Somma(n,m) << endl;
 n=2; m=2;
 cout << " n = " << n << " m = " << m << " Somma(n,m) = " <<
  Somma(n,m) << endl;
 system("pause");system("CLS");

 cout << "Prova pal(s)" << endl;
 s="osso";
 cout << endl << " s       = " << s << endl << " pal(s) = " <<
pal(s) << endl;
 s="radar";
 cout << endl << " s       = " << s << endl << " pal(s) = " <<
pal(s) << endl;
 s="kajak";
 cout << endl << " s       = " << s << endl << " pal(s) = " <<
pal(s) << endl;
 s="ailatiditalia";
 cout << endl << " s       = " << s << endl << " pal(s) = " <<
pal(s) << endl;
 s="omordotuanuoraoarounautodromo";
 cout << endl << " s       = " << s << endl << " pal(s) = " <<
pal(s) << endl;
```

```
 s="asso";
 cout << endl << " s        = " << s << endl << " pal(s) = " <<
pal(s) << endl;
 s="orso";
 cout << endl << " s        = " << s << endl << " pal(s) = " <<
pal(s) << endl;
 system("pause");system("CLS");

 cout << "Prova contenuta(s1,s2)" << endl;
 s1="NERO"; s2="GANTTEORO";
 cout << " s1 = " << s1 << " s2 = " << s2 << " contenuta(s1,s2) =
" <<
  contenuta(s1,s2) << endl;
 s1="NERO"; s2="GANTTEOO";
 cout << " s1 = " << s1 << " s2 = " << s2 << " contenuta(s1,s2) =
" <<
  contenuta(s1,s2) << endl;
 system("pause");system("CLS");}
```

Settimana 08 – Strutture
"Insiemi di dati composti"

1. **Lezione 15: Strutture in C++ (capitolo 8 libro testo)**
2. **Lezione 16: Strutture in C++ (capitolo 9 libro testo)**
3. **Tutorato 08: Strutture in C++**



Un esempio: possiamo formare l'insieme delle coppie di dati presi da due insiemi di dati già esistenti

Lezione 15 - Strutture: i punti
Parte 1. Passaggio per valore e referenza
Esempi (Cap. 8 libro testo)

In questa lezione vediamo come sia possibile definire tipi detti ***strutture***, che rappresentano nuovi ***insiemi di dati***. Una struttura è l'insieme di tutte le possibili liste che possiamo ottenere una volta scelti: la ***lunghezza*** della lista, il ***tipo*** di ogni elemento della lista, e il ***"nome"*** di ogni elemento. Un esempio. Definiamo la struttura Point come l'insieme: $\{ \{p_x, p_y\} \mid p_x, p_y \text{ numeri reali}\}$ di tutte le liste di due elementi, entrambi numeri reali, in cui chiamiamo il primo elemento ***"componente x"***, e il secondo ***"componente y"***. Point rappresenta l'insieme dei punti del piano. La definizione di Point in C++ è

```
struct Point {double x, y;};
```

Dato p in Point, la componente x si indica in C++ con p.x e la componente y con p.y: se Origine = {0.0, 0.0}, allora Origine.x = 0 e Origine.y = 0. Vediamo ora i dettagli.

1. **(Capitolo 8.3)** Le variabili di tipo Point si dichiarano come per un qualunque altro tipo, scrivendo Point p, q; Per assegnare a p un valore dobbiamo assegnare un valore a ogni componente di p, scrivendo p.x = 3.0; p.y = 4.0. Sono possibili abbreviazioni come p = (Point) {3.0, 4.0}; oppure dichiarazioni fatte insieme ad assegnazioni come Point p = {3.0, 4.0}. Se p, q hanno tipo Point si può scrivere p = q;
2. **(Capitolo 8.5)** Gli elementi di una struttura possono essere valori in ingresso e valori in uscita delle funzioni che definiamo.
3. **(Capitoli 8.8, 8.9)** Una struttura può essere definita a partire da altre strutture. Vedremo come definire una struttura Rectangle (l'insieme dei rettangoli del piano) a partire dalla struttura Point (l'insieme dei punti del piano).
4. **(Capitolo 8.7, 8.10)** Una funzione può passare i propri argomenti ***per valore*** oppure ***per referenza***. Se non chiediamo esplicitamente il contrario, ogni argomento viene passato ***per valore***. In questo caso, la funzione riceve una ***copia*** dell'argomento e ***eventuali modifiche fatte sull'argomento svaniscono*** non appena l'esecuzione della funzione è terminata. Nelle funzioni definite su strutture, per evitare di duplicare grandi aree di memoria, si utilizza spesso il ***"passaggio per referenza"***. Quando un argomento viene passato per referenza, ogni modifica che la funzione esegue sul proprio argomento ***avviene sul valore originario dell'argomento***, e resta quando l'esecuzione della

funzione è terminata. Per indicare che un argomento di una funzione viene passato per referenza dobbiamo inserirci una & davanti nella definizione della funzione. Un passaggio per referenza comporta il rischio di modificare un dato che ci servirebbe ancora, ma a volte è indispensabile. Come esempio, vedremo la funzione **void** reflect2 (Point& p); che scambia le componenti x e y del punto p. Questa funzione, come vedremo, si può scrivere solo usando il passaggio per referenza.

```
// Lezione15-Strutture. PRIMA ORA.
#include <iostream>
#include <stdlib.h>
#include <math.h>
using namespace std;

//ELENCO DELLE FUNZIONI E STRUTTURE CHE PRESENTEREMO OGGI
//Ne spiegheremo l'uso man mano che si presentano nel main.

//Struttura Point che rappresenta l'insieme dei punti del piano
struct Point {double x, y;}; //non dobbiamo dimenticare il ;

void printPoint (Point p) ;
double d (Point p1, Point p2) ;
void reflect (Point p);
void reflect2 (Point& p);

/* Struttura Rectangle che rappresenta l'insieme dei rettangoli
del piano.
 Per individuare un rettangolo è sufficiente conoscere il suo
angolo inferiore sinistro e base, altezza: tutte le altre
informazioni possono venire dedotte. */
struct Rectangle
{Point corner;          //angolo inferiore sinistro
 double width, height;  //base, altezza
}; //non dobbiamo dimenticare il ;

Point findCenter (Rectangle& box);
void swap (int& x, int& y);

int main()
{
 Point blank;
 blank.x = 3.0;
 blank.y = 4.0;
```

```
/* Capitolo 8.3. Spieghiamo come dichiara e assegnare variabili di
tipo Point */
 cout << "Capitolo 8.3" << endl <<
  "Come accedere a parti di una struttura" << endl << endl;
 cout << " punto blank = " << blank.x << ", " << blank.y << endl;
 double sqdist = blank.x * blank.x + blank.y * blank.y;
 cout << " quadrato distanza dall'origine = " << sqdist << endl;
 Point blank2; Point p1 = { 3.0, 4.0 };
 //se assegno {a,b} devo precisare che {a, b} ha tipo Point
 blank2 = (Point) { 0.0, 0.0 };
 //se assegno p1 variabile di tipo Point non devo precisare nulla
 Point p2 = p1;
 cout << " Stampo il punto p2" << endl;
 cout << " "; printPoint(p2);
 cout << endl; system("pause"); system("CLS");

/* Capitolo 8.5. Possiamo definire funzioni con elementi di tipo
Point come argomenti e come valore di ritorno. */
 cout << "Capitolo 8.5" << endl << "Gli elementi di una struttura
possono essere argomenti di funzioni" << endl << endl;
 cout << " calcolo distanza blank, blank2" << endl;
 cout << " d(blank,blank2) = " << d(blank,blank2) << endl;
 cout << endl; system("pause");system("CLS");

/* Capitolo 8.7. A volte è necessario che le strutture vengono
passate "per referenza". Per esempio, la funzione che scambia le
componenti x e y di un punto si può scrivere solo usando il
passaggio per referenza. */
 cout << "Capitolo 8.7" << endl <<
  "Passaggio per referenza" << endl << endl;
 cout << " Stampo il punto p2" << endl;
 cout << " "; printPoint(p2);
 cout << " applico reflect (p2): il punto non cambia" << endl;
 reflect(p2); cout << " "; printPoint(p2);
 cout << " Questo perche' reflect2 non usa il passaggio per
referenza" << endl;
 cout << " Applico reflect2(p2): grazie al passaggio per referenza
il punto cambia" << endl;
 reflect2(p2); cout << " "; printPoint(p2);
 cout << endl; system("pause");system("CLS");

/* Capitolo 8.8, 8.9. Possiamo definire strutture a partire da
altre strutture. */
 cout << "Capitolo 8.8" << endl <<
```

```
  "Un esempio di strutture annidate: i rettangoli" << endl <<
endl;
 Point corner = { 0.0, 0.0 };
 //un rettangolo è determinato noti il margine inferiore sinistro
 //una base e una altezza. Le altre informazioni sono deducibili.
 Rectangle box  = { corner, 100.0, 200.0 };
 //margine inferiore sinistro = corner, base=100, altezza=200
 //una assegnazione equivalente sarebbe
 Rectangle box2 = { { 0.0, 0.0 }, 100.0, 200.0 };
 cout << " angolo inferiore sinistro box= "; printPoint(corner);
 cout << " box.width                    = " << box.width << endl;
 cout << " box.height                   = " << box.height << endl;
 //posso cambiare ogni singolo dato del rettangolo
 box.width = box.width + 50.0;
 //la base ora è di 50 unita' piu' grande
 cout << " nuova box.width              = " << box.width << endl;
 Point temp = box.corner;
 double x = box.corner.x;
 //in una struttura annidata una componente box.corner puo' avere
componenti
 cout << " box.corner.x                  = " << x << endl;
 cout << endl; system("pause");system("CLS");

 cout << "Capitolo 8.9" << endl <<
  "Una funzione che calcola il baricentro di un rettangolo"
  << endl << endl;
 Point center = findCenter (box);
 cout << " baricentro box = ";
 printPoint (center);
 cout << endl; system("pause");system("CLS");

 /* Capitolo 8.10. Un altro esempio di passaggio per referenza */
 cout << "Capitolo 8.10" << endl <<
  "passaggio per referenza di interi" << endl << endl;
 int i = 7, j = 9;
 cout << " Prima dell'esecuzione di swap(i,j)" << endl;
 cout << " i = " << i << " j = " << j << endl;
 swap (i, j);
 cout  <<  "  Dopo  l'esecuzione  di  swap(i,j)  gli  interi  sono
scambiati" << endl;
 cout << " i = " << i << " j = " << j << endl;
 cout << endl; system("pause");system("CLS");
}

//Stampa delle coordinate di un punto
```

```
void printPoint (Point p)
{cout << "(" << p.x << ", " << p.y << ")" << endl;}

//Distanza tra due punti
double d (Point p1, Point p2)
{double dx = p2.x - p1.x;
 double dy = p2.y - p1.y;
 return sqrt (dx*dx + dy*dy);}

/* Esempio di funzione "reflect" che modifica l'argomento,
scambiando le coordinate di un punto P. "reflect" riflette P
rispetto alla diagonale principale. Per esempio, P'={1,2} e' la
riflessione di P={2,1}:

Y   (diagonale principale)
|    /
| * P' = {1,2}
|  /
| /  * P = {2,1}
|/_____________________X

*/
void reflect (Point p)
/* VERSIONE ERRATA di reflect. Le funzioni creano copie temporanee
dei loro argomenti per evitare di modificarli accidentalmente.
Come conseguenza, questa versione di reflect non modifica il punto
p originario, ma solo una copia temporanea che va persa. */
{double temp = p.x; //salvo una copia di p.x
 p.x = p.y;
 p.y = temp;}

/* VERSIONE CORRETTA di reflect. Il simbolo & fa si che la
funzione agisca sul punto p originario e non su una copia di p.
Questo modo di trattare gli argomenti viene detto passaggio per
referenza. */
void reflect2 (Point& p)
{double temp = p.x; //salvo una copia di p.x
 p.x = p.y;
 p.y = temp;}

//funzione che calcola il baricentro di un rettangolo
Point findCenter (Rectangle& box)
{double x = box.corner.x + box.width/2;
 double y = box.corner.y + box.height/2;
 Point result = {x, y};  return result;}
```

```
/* Funzione swap che scambia due interi. La funzione utilizza il
simbolo & per agire sui valori originari di x,y, e non su loro
copie temporanee. */
void swap (int& x, int& y)
{int x_salvato = x;// prima di assegnare y=x salvo una copia di x
 x = y;            // ora x non contiene piu' il valore originario
 y = x_salvato;    // assegno la copia salvata di x a y
}
```

Lezione 15 - Strutture – Punti
Parte 2. Esercizi

Per rappresentare i punti del piano utilizziamo la struttura Point:

struct Point {double x, y;};

vista a lezione. Per stampare un punto usate:

```
void printPoint (Point p)
 {cout << "(" << p.x << ", " << p.y << ")" << endl;}
```

Es 15.1 (vettore da P1 a P2). Definire una funzione:

Point vett (Point P1, Point P2)

che prende due punti **P1={x1,y1}**, **P2={x2,y2}** e restituisce il vettore da P1 a P2, rappresentato dal punto di coordinate **{x2-x1, y2-y1}**. **Esempio.** Siano P1={1,2}, P2={4,6}. Controllate che vett(P1,P2)={4-1,6-2}={3,4}.

Es 15.2 (distanza da P1 a P2). Definire una funzione:

double dist(Point P1, Point P2)

che prende due punti P1={x1,y1}, P2={x2,y2} e restituisce la distanza da P1 a P2, calcolata con $\sqrt{((x2-x1)^2 + (y2-y1)^2)}$. **Esempio.** Siano P1={1,2}, P2={4,6}. Controllate che dist(P1,P2) = $\sqrt{(3^2+4^2)}$= $\sqrt{25}$ = 5.

Es 15.3 (prodotto scalare di V1 e V2). Definire una funzione:

double scal(Point V1, Point V2)

che prende due vettori V1={x1,y1}, V2={x2,y2} (rappresentati con il loro punto di arrivo) e restituisce il prodotto scalare di V1 e V2, calcolato con la formula (x1*x2+y1*y2). **Esempio.** Siano P1={1,2}, P2={4,6}. Controllate che scal(P1,P2) = 1*4 + 2*6 = 16.

Es 15.4 (media pesata di P1, P2,P3). *Per la definizione di "media pesata" di numeri rimandiamo a Wikipedia.* Definiamo la "media pesata" di tre punti P1,P2,P3 come il punto che ha come coordinate le "medie pesate" delle coordinate. Definite una funzione:

Point media(Point P1, Point P2, Point P3,
double p1, double p2, double p3)

che prende tre punti P1={x1,y1}, P2={x2,y2}, P3={x3,y3}, tre reali p1, p2, p3 di somma non nulla, detti "pesi" e restituisce la "media pesata" di P1,P2,P3, calcolata come segue:

```
{ (x1*p1 + x2*p2 + x3*p3)/(p1+p2+p3),
  (y1*p1 + y2*p2 + y3*p3)/(p1+p2+p3) }
```

Esempio. Siano P1={0,0}, P2={1,0}, P3={0,1}. Controllate che sia media(P1,P2,P3,1,1,1) = {1/3,1/3}.

Lezione 15 – Strutture: Punti
Parte 3. Soluzioni

```
// Lezione15-Strutture-Punti-Soluzioni
#include <iostream>
#include <stdlib.h>
#include <math.h>
using namespace std;

struct Point {double x, y;}; //non dimenticate il ;

void printPoint (Point p)
{cout << "(" << p.x << ", " << p.y << ")" << endl;}

Point vett (Point P1, Point P2)
{Point V = {P2.x - P1.x, P2.y - P1.y};
 return V;}

double dist (Point p1, Point p2)
{
 double dx = p2.x - p1.x;
 double dy = p2.y - p1.y;
 double d  = sqrt (dx*dx + dy*dy);
 return d;
}

double scal(Point V1, Point V2)
{ double prodscal = (V1.x) * (V2.x) + (V1.y) * (V2.y);
  return prodscal;}

Point media(Point P1, Point P2, Point P3,
double p1, double p2, double p3)
//si richiede che la somma dei pesi sia diversa da 0
{double pesi = p1 + p2 + p3;
 Point P = {((P1.x)*p1 + (P2.x)*p2 + (P3.x)*p3)/pesi,
            ((P1.y)*p1 + (P2.y)*p2 + (P3.y)*p3)/pesi};
 return P;
}

int main()
{ Point P1={1,2}, P2={4,6}, P3;
  cout << "Test delle funzioni: vett, dist, scal, media"
   << endl << endl;
  cout << " P1                    = "; printPoint(P1);
  cout << " P2                    = "; printPoint(P2);
```

```
cout << " vett(P1,P2)           = "; printPoint(vett(P1,P2));
cout << " dist(P1,P2)           = " << dist(P1,P2) << endl;
cout << " scal(P1,P2)           = " << scal(P1,P2) << endl;
P1 = (Point) {0.0, 0.0};
cout << " P1                    = "; printPoint(P1);
P2 = (Point) {1.0, 0.0};
cout << " P2                    = "; printPoint(P2);
P3 = (Point) {0.0, 1.0};
cout << " P3                    = "; printPoint(P3);
cout << " media(P1,P2,P3,1,1,1) = ";
printPoint(media(P1,P2,P3,1,1,1));
cout << endl; system("pause"); system("CLS");}
```

Lezione 16 - Strutture
Parte 1. Esempi (Cap. 9)

In questa lezione continuiamo a fornire esempi di strutture e di funzioni definite su di esse. L'esempio scelto dal libro di testo (Capitolo 9) è una struttura Time che rappresenta il tempo trascorso a partire da qualche istante scelto come iniziale, e scomposto in ore (qualunque numero intero), minuti (solo da 0 a 59), secondi (solo da 0 a 59). Su questa struttura vengono definite le operazioni: stampa di un elemento di Time, confronto e somma di due elementi di Time, conversione di un elemento di Time in secondi, aggiunta di un numero di secondi a un elemento di Time.

Ricordiamo che indicare un argomento di una funzione come "Time& t" anziché "Time t" consente a una funzione di operare sul valore originario dell'argomento *(passaggio per referenza)* anziché su una copia *(passaggio per valore)*. Questo modo di usare un argomento consente a una funzione di modificare il suo argomento e diminuisce le copie non necessarie, ma aumenta anche le possibilità di errore.

```
// Lezione16-Strutture-Ora
#include <iostream>
#include <stdlib.h>
#include <math.h>
using namespace std;

/* Tempo scomposto in ore (qualunque numero), minuti (da 0 a 59),
secondi (da 0 a 59)*/
struct Time
{int hour,minute; double second;}; //non dobbiamo dimenticare il ;

/* Cap. 9.2. stampa di un elemento di Time */
void printTime (Time& t);
/* Cap. 9.4. Confronto tra due elementi di Time: se time1 viene
dopo di time2 risponde vero */
bool after (Time& time1, Time& time2);
/* Cap. 9.4. somma tra due elementi di Time: la prima versione è
errata perché non tiene conto del possible riporto di 1 */
Time addTime (Time& t1, Time& t2);
Time addTime2 (Time& t1, Time& t2);

/* Cap. 9.5. versione di printTime con argomento dichiarato
"costante", ovvero non modificabile. Dichiarare costante un valore
serve a evitare modifiche accidentali. */
```

```
void printTime2 (const Time& t);

/* Cap. 9.6. Funzione incremento, che modifica il proprio
argomento. La prima versione è errata perché considera solo la
possibilità di un riporto di 1 e non riporti più grandi. */
void increment  (Time& time, double secs) ;
void increment2 (Time& time, double secs) ;

/* Un altro modo di definire una somma è non restituire un
risultato, ma scrivere il risultato modificando l'ultimo degli
argomenti. I primi due argomenti vengono dichiarati "costanti" per
evitare modifiche accidentali. */
void addTimeFill (const Time& t1, const Time& t2, Time& sum);
//Conversione della misura del tempo in secondi
double convertToSeconds (const Time& t);
//Conversione opposta
Time makeTime (double secs) ;
// Somma di due tempi usando la conversione in secondi
Time addTime3 (const Time& t1, const Time& t2) ;

int main()
{cout << "Cap. 9.2: stampa dell'ora" << endl;
 Time time = { 11, 59, 3.1 };
 cout << "time = "; printTime(time);
 system("pause");system("CLS");

 cout << "Cap. 9.4: funzioni pure (che non modificano il loro
argomento)" << endl;
 Time currentTime = { 9, 14, 30.0 };
 Time breadTime   = { 3, 45, 30.0 };
 cout << " currentTime                 = ";printTime(currentTime);
 cout << " breadTime                   = ";printTime(breadTime);
 Time doneTime  = addTime (currentTime, breadTime);
 cout << " doneTime                    = ";printTime (doneTime);
 Time doneTime2 = addTime2 (currentTime, breadTime);
 cout << " doneTime2                   = ";printTime (doneTime2);
 cout << " after(doneTime,currentTime) = "
  << after(doneTime,currentTime) << endl;
 system("pause");system("CLS");

 cout << "Cap. 9.5: uso di const" << endl;
/*quando si dichiara un argomento di una funzione come "const" non
si puo' modificarlo. const serve ad evitare modifiche accidentali
e non produce effetti visibili sull'esecuzione del programma */
 cout << " time = "; printTime(time);
```

```
 system("pause");system("CLS");

 cout << "Cap. 9.6: funzioni che modificano il loro argomento"
  << endl; // ricordiamo che currentTime = { 9, 14, 30.0 }
 cout << " currentTime                  = ";printTime(currentTime);
 cout << " applico increment2(currentTime,10000);" << endl;
 increment2(currentTime,10000);
 cout << " currentTime                  = ";printTime(currentTime);
 system("pause");system("CLS");}


//DEFINIZIONE DELLE FUNZIONI USATE NELLA LEZIONE 16
void printTime (Time& t)
{cout << t.hour << ":" << t.minute << ":" << t.second << endl;}

bool after (Time& time1, Time& time2)
{if (time1.hour > time2.hour)     return true;
 if (time1.hour < time2.hour)     return false;
 if (time1.minute > time2.minute) return true;
 if (time1.minute < time2.minute) return false;
 if (time1.second > time2.second) return true;
//se non vale nessuno dei casi precedenti
 return false;}

//una versione di addTIme che non tiene conto dei riporti
Time addTime (Time& t1, Time& t2)
{Time sum;
 sum.hour = t1.hour + t2.hour;
 sum.minute = t1.minute + t2.minute;
 sum.second = t1.second + t2.second;
//quando la somma supera i 60 secondi il risultato è errato
 return sum;}

//una versione corretta che tiene conto di un riporto di 1
Time addTime2 (Time& t1, Time& t2)
{Time sum;
  sum.hour = t1.hour + t2.hour;
  sum.minute = t1.minute + t2.minute;
  sum.second = t1.second + t2.second;
  if (sum.second >= 60.0)
   {sum.second -= 60.0;
    sum.minute += 1;}
 if (sum.minute >= 60)
  {sum.minute -= 60;
   sum.hour += 1;}
```

```
 return sum;}

//versione di printTime con argomento costante
void printTime2 (const Time& t)
{cout << t.hour << ":" << t.minute << ":" <<
   t.second << endl;}

/* funzione incremento, con riporto di 1: quando il riporto è
maggiore di 1 non è corretta. */
void increment (Time& time, double secs)
{time.second += secs;
 if (time.second >= 60.0)
 {time.second -= 60.0;
  time.minute += 1;}
if (time.minute >= 60)
 {time.minute -= 60;
  time.hour += 1;}
}

// funzione incremento, che tiene conto di un riporto illimitato
void increment2 (Time& time, double secs)
{time.second += secs;
 while (time.second >= 60.0)
 {time.second -= 60.0;
  time.minute += 1;}
while (time.minute >= 60)
 {time.minute -= 60;
  time.hour += 1;}
}

/* Un altro modo di definire una somma è non restituire un
risultato, ma scrivere il risultato modificando uno degli
argomenti */
void addTimeFill (const Time& t1, const Time& t2, Time& sum)
{sum.hour = t1.hour + t2.hour;
 sum.minute = t1.minute + t2.minute;
 sum.second = t1.second + t2.second;
 if (sum.second >= 60.0)
  {sum.second -= 60.0;
   sum.minute += 1;}
 if (sum.minute >= 60)
  {sum.minute -= 60;
   sum.hour += 1;}
}

//Conversione della misura del tempo in secondi
double convertToSeconds (const Time& t)
```

```
{int minutes = t.hour * 60 + t.minute;
 double seconds = minutes * 60 + t.second;
 return seconds;}

//Conversione opposta
Time makeTime (double secs)
{Time time;
 time.hour = int (secs / 3600.0);
 secs -= time.hour * 3600.0;
 time.minute = int (secs / 60.0);
 secs -= time.minute * 60;
 time.second = secs;
 return time;}

/* Per sommare due tempi, senza doversi preoccupare dei riporti,
il metodo piu' semplice consiste nel convertire i tempi in
secondi, sommarli come interi e riconvertire in ore, minuti e
secondi il risultato */
Time addTime3 (const Time& t1, const Time& t2) {
double seconds = convertToSeconds (t1) + convertToSeconds (t2);
return makeTime (seconds);}
```

Lezione 16: Strutture
Parte 2. Esercizi

In questa esercitazione calcoleremo i luoghi notevoli di un triangolo come particolari medie pesate dei vertici. Nella soluzione, vi chiediamo di riutilizzare la struttura

struct Point {double x, y;};

e le funzioni

Point vett (Point P1, Point P2)
double dist(Point P1, Point P2)
double scal(Point V1, Point V2)
Point media(Point P1, Point P2, Point P3,
double p1, double p2, double p3)

date come esercizi nella Lezione 15.

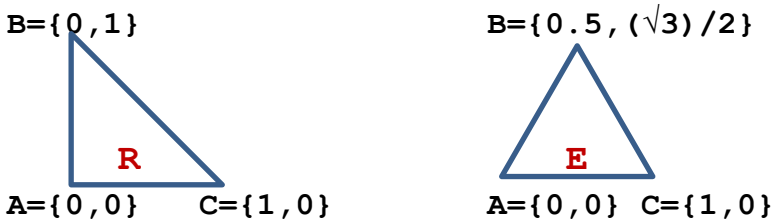
Es 16.1 (triangoli) Definite una struttura **Triangolo**, che descrive un triangolo dati in senso orario i tre vertici A,B,C, rappresentati con elementi di Point. Non dimenticate il punto e virgola alle fine della definizione. Quindi definite due triangoli: il triangolo rettangolo isoscele:

R = { {0.0, 0.0}, {0.0,1.0}, {1.0, 0.0} }

e il triangolo equilatero

E = { {0.0, 0.0}, {0.5, (√3)/2}, {1.0, 0.0} }

La radice si calcola con la funzione sqrt(.). I due triangoli sono:



Es 16.2 (baricentro di un triangolo). Definite una funzione:

Point baricentro(Triangolo T)

che prende un triangolo T = {P1, P2, P3} e ne restituisce il baricentro, calcolato come la media pesata dei vertici di pesi

p1 = 1
p2 = 1
p3 = 1

Esempi. Controllate che baricentro(R) = {1/3,1/3} e che baricentro(E) = {1/2,(√3)/6}, con (√3)/6 = 0.288 … .

Es 16.3 (circocentro di un triangolo). Definite una funzione:

Point circocentro(Triangolo T)

che prende un triangolo T = {P1, P2, P3} e ne restituisce il circocentro, calcolato come la media pesata di P1, P2, P3 di pesi:
p1 = scal(vett(P1,P2),vett(P1,P3)) * scal(vett(P2,P3),vett(P2,P3))
p2 = scal(vett(P2,P3),vett(P2,P1)) * scal(vett(P1,P3),vett(P1,P3))
p3 = scal(vett(P3,P1),vett(P3,P2)) * scal(vett(P1,P2),vett(P1,P2))
Esempi. Controllate che circocentro(R) = {1/2,1/2} e che circocentro(E) = {1/2,(√3)/6}, con (√3)/6 = 0.288 … .

Es 16.4 (incentro di un triangolo). Definire una funzione:
Point incentro(Triangolo T)
che prende un triangolo T = {P1, P2, P3} e ne restituisce l'incentro, calcolato come la media pesata di P1, P2, P3 di pesi
p1 = dist(P2,P3)
p2 = dist(P1,P3)
p3 = dist(P1,P2)
Esempi. Controllate che incentro(R) = {1-1/√2,1-1/√2}, con 1-1/√2 = 0.292 …, e che incentro(E) = {1/2,(√3)/6}, con (√3)/6 = 0.288 ….

Es 16.5 (ortocentro di un triangolo). Definite una funzione:
Point ortocentro(Triangolo T)
che prende un triangolo T = {P1, P2, P3} e ne restituisce l'ortocentro, calcolato come la media pesata di P1, P2, P3 di pesi
p1 = scal(vett(P1,P2),vett(P2,P3)) * scal(vett(P2,P3),vett(P3,P1))
p2 = scal(vett(P2,P3),vett(P3,P1)) * scal(vett(P3,P1),vett(P1,P2))
p3 = scal(vett(P3,P1),vett(P1,P2)) * scal(vett(P1,P2),vett(P2,P3))
Esempi. Controllate che ortocentro(R) = {0,0}, e che ortocentro(E) = {1/2,(√3)/6}, con (√3)/6 = 0.288 ….

Lezione 16 - Strutture
Parte 3. Soluzioni

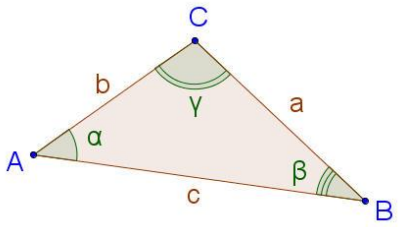
```
// Lezione16-Strutture-Punti-Soluzioni
#include <iostream>
#include <stdlib.h>
#include <math.h>
using namespace std;

struct Point {double x, y;}; //non dimenticate il ;

void printPoint (Point p)
{cout << "(" << p.x << ", " << p.y << ")" << endl;}

Point vett (Point P1, Point P2)
{Point V = {P2.x - P1.x, P2.y - P1.y};
 return V;}

double dist (Point p1, Point p2)
{
 double dx = p2.x - p1.x;
 double dy = p2.y - p1.y;
 double d  = sqrt (dx*dx + dy*dy);
 return d;
}

double scal(Point V1, Point V2)
{ double prodscal = (V1.x) * (V2.x) + (V1.y) * (V2.y);
  return prodscal;}

Point media(Point P1, Point P2, Point P3,
double p1, double p2, double p3)
{double pesi = p1 + p2 + p3;
 Point P = {((P1.x)*p1 + (P2.x)*p2 + (P3.x)*p3)/pesi,
            ((P1.y)*p1 + (P2.y)*p2 + (P3.y)*p3)/pesi};
 return P;
}

struct Triangolo
{Point A, B, C;};

Point baricentro(Triangolo T)
{Point P1=T.A, P2=T.B, P3=T.C;
 double p1 = 1;
```

```
 double p2 = 1;
 double p3 = 1;
 return media(P1,P2,P3,p1,p2,p3);}

Point circocentro(Triangolo T)
{Point P1=T.A, P2=T.B, P3=T.C;
 double       p1       =       scal(vett(P1,P2),vett(P1,P3))       *
scal(vett(P2,P3),vett(P2,P3));
 double       p2       =       scal(vett(P2,P3),vett(P2,P1))       *
scal(vett(P1,P3),vett(P1,P3));
 double       p3       =       scal(vett(P3,P1),vett(P3,P2))       *
scal(vett(P1,P2),vett(P1,P2));
 return media(P1,P2,P3,p1,p2,p3);}

Point incentro(Triangolo T)
{Point P1=T.A, P2=T.B, P3=T.C;
 double p1 = dist(P2,P3);
 double p2 = dist(P1,P3);
 double p3 = dist(P1,P2);
 return media(P1,P2,P3,p1,p2,p3);}

Point ortocentro(Triangolo T)
{Point P1=T.A, P2=T.B, P3=T.C;
 double       p1       =       scal(vett(P1,P2),vett(P2,P3))       *
scal(vett(P2,P3),vett(P3,P1));
 double       p2       =       scal(vett(P2,P3),vett(P3,P1))       *
scal(vett(P3,P1),vett(P1,P2));
 double       p3       =       scal(vett(P3,P1),vett(P1,P2))       *
scal(vett(P1,P2),vett(P2,P3));
 return media(P1,P2,P3,p1,p2,p3);}

int main()
{ Triangolo R = { {0.0, 0.0}, {0.0,1.0}, {1.0, 0.0} };
  Triangolo E = { {0.0, 0.0}, {0.5, sqrt(3)/2}, {1.0, 0.0} };
  cout << "Test delle funzioni:" << endl <<
   "baricentro, circocentro, incentro, ortocentro" << endl;
  cout << " Triangolo R:" << endl;
  cout << " ";printPoint(R.A);
  cout << " ";printPoint(R.B);
  cout << " ";printPoint(R.C);
  cout << " Triangolo E:" << endl;
  cout << " ";printPoint(E.A);
  cout << " ";printPoint(E.B);
  cout << " ";printPoint(E.C);
  cout << " baricentro(R)  = "; printPoint(baricentro(R));
```

```
cout << " baricentro(E)  = "; printPoint(baricentro(E));
cout << " circocentro(R) = "; printPoint(circocentro(R));
cout << " circocentro(E) = "; printPoint(circocentro(E));
cout << " incentro(R)    = "; printPoint(incentro(R));
cout << " incentro(E)    = "; printPoint(incentro(E));
cout << " ortocentro(R)  = "; printPoint(ortocentro(R));
cout << " ortocentro(E)  = "; printPoint(ortocentro(E));
system("pause"); system("CLS");}
```

Tutorato 08 - Strutture in C++
Parte 1. Esercizi proposti

La struttura Triangolo. Definire una struttura Triangolo che rappresenta un triangolo T a partire da tre numeri reali: le lunghezze a, b, c dei lati opposti ai vertici A, B, C. Disegnamo A,B,C in ordine ***antiorario*** partendo in basso a sinistra, e chiamiamo un angolo in base al suo vertice e un lato in base al vertice opposto:



Nota 1. Questa struttura Triangolo è diversa dalla struttura con lo stesso nome vista nella Lezione 16, in cui un triangolo veniva rappresentato da 6 numeri, le coordinate dei suoi tre vertici.
Nota 2. Siano date le lunghezze a, b, c dei lati di T. Allora l'area di T si calcola con la ***formula di Erone***. Gli angoli α, β, γ ai vertici A, B, C di T si calcolano con il ***Teorema di Carnot o del Coseno*** (sono formule che trovate su Wikipedia).

Tut 08.1 Scrivere una funzione

```
void stampaTriangolo(Triangolo  T);
```

che prende un triangolo T e ne stampa i tre lati.

Tut 08.2 Scrivere una funzione

```
void inserisciTriangolo(Triangolo&  T);
```

che prende una variabile T di tipo triangolo e richiede all'utente di inserire i tre lati da tastiera, quindi li stampa per verifica, usando la funzione precedente. Il passaggio per riferimento della variabile T, indicato dal simbolo &, è necessario per modificare il contenuto della variabile T originaria e non agire su una copia.

Tut 08.3 Scrivere una funzione

```
bool triangolo(Triangolo T);
```

che verifichi se i 3 lati inseriti formano effettivamente un triangolo. Tre lati formano un triangolo se e solo se ognuno di essi è minore della somma degli altri due.

Tut 08.4 Scrivere una funzione

```
void stampa_angoli(Triangolo T);
```

che stampi l'ampiezza in radianti di tutti gli angoli del triangolo.

Tut 08.5 Scrivere una funzione

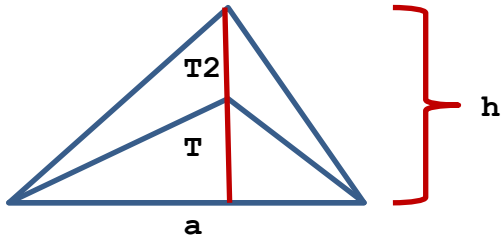
```
double altezza_a(Triangolo T);
```

che calcoli l'altezza del triangolo considerando il lato a come base.

Tut 08.6 Scrivere una funzione

```
Triangolo cambia_altezza_a(Triangolo T, double h)
```

che che produca un nuovo triangolo T2 di altezza h rispetto al lato a. T2 è ottenuto dilatando o contraendo l'altezza di T rispetto al lato a, mantenendola lungo la stessa retta, e cambiando le lunghezze dei lati b e c di conseguenza.



Esempi per Tut08.5 e Tut08.6. Verificare che dato un triangolo di lati a=40, b=30, c=20, l'altezza relativa alla base a sia 14.52. Inoltre verificare che con gli stessi valori data una nuova altezza h = 20 i nuovi lati siano a=40, b=33.0009, c=24.2706.

Tutorato 08 – strutture in C++
Parte 2. Soluzioni

```
// Tutorato 08 - Soluzioni
#include <iostream>
#include <string>
#include <stdlib.h>
#include <math.h>
using namespace std;

struct Triangolo {double lato_a, lato_b, lato_c;};
//non dimenticate il ;

//Tut 08.1. Stampa dei dati relativi al triangolo T
void stampaTriangolo(Triangolo T)
{
 cout << "Lati del triangolo: \n a = " << T.lato_a << " (la base)"
<< "\n b  = "
  << T.lato_b << "\n c = " << T.lato_c << endl;
}

//Tut 08.2. Inserimento dei dati relativi al triangolo T
void inserisciTriangolo(Triangolo& T)
//Uso il passaggio per riferimento per poter modificare T
{
 cout << "Inserire i 3 lati del triangolo" << endl;
 cout << "lato a (la base del triangolo)" << endl;
 cin >> T.lato_a;
 cout << "lato b" << endl;
 cin >> T.lato_b;
 cout << "lato c" << endl;
 cin >> T.lato_c;
 stampaTriangolo(T);
}

//Tut 08.3 Controllo se T e' un triangolo
bool triangolo (Triangolo T)
{double a = T.lato_a, b = T.lato_b, c = T.lato_c;
 bool condizione_per_triangolo =
  (a < b+c ) && (b < a+c ) && (c < a+b);
 return condizione_per_triangolo;
}
```

```
//Tut 08.4. Stampa degli angoli del triangolo T
void stampa_angoli(Triangolo T)
{double a = T.lato_a, b = T.lato_b, c = T.lato_c;
// cos (alfa) = (b*b+c*c-a*a)/(2*b*c)
 cout << "angolo alfa \t = radianti ";
 cout << acos((b*b+c*c-a*a )/(2*b*c)) << endl;
// cos (beta) = (a*a+c*c-b*b)/(2*a*c)
 cout << "angolo beta \t = radianti ";
 cout << acos((a*a+c*c-b*b)/(2*a*c)) << endl;
// cos (gamma) = (a*a+b*b-c*c)/(2*a*b)
 cout << "angolo gamma \t = radianti ";
 cout << acos((a*a+b*b-c*c)/(2*a*b)) << endl;
}

//Tut 08.5. Calcolo dell'altezza di T relativa al lato a
double altezza_a(Triangolo T)
{double a = T.lato_a, b = T.lato_b, c = T.lato_c;
 double base = a;
// calcoliamo il semiperimetro p
 double p = (a+b+c)/2;
// calcoliamo l'area
 double area = sqrt(p*(p-a)*(p-b)*(p-c));
// area = base*altezza/2
 double altezza = 2*area/base;
 return altezza;
}

//Tut 08.6. Nuovo triangolo di altezza h relativa al lato a
Triangolo cambia_altezza_a(Triangolo T, double h)
{double a = T.lato_a, b = T.lato_b, c = T.lato_c;
 double altezza = altezza_a(T); //altezza originaria rispetto al
lato a
 double pb, pc; //proiezioni di b e c su a
 pb = sqrt(b*b - altezza*altezza);
 pc = sqrt(c*c - altezza*altezza);
//ricavo i nuovi lati b e c con il teorema di pitagora
 T.lato_b = sqrt(pb*pb + h*h);
 T.lato_c = sqrt(pc*pc + h*h);
 return T;
}

//funzione utilizzata per stampare un titolo
void titolo(string s)
{string riga = "----------------------------";
```

```
 cout << riga << endl << s << endl << riga << endl; }


int main() {char e_accentata = '\212'; //codice ASCII per "e
accentata"

 titolo(" Tut 08.1, 08.2. La struttura Triangolo.");
// inizializzazione triangolo
 Triangolo T; inserisciTriangolo(T);

// Tut 08.3. Controllo se il nuovo T è un triangolo
  titolo(" Tut 08.3."); bool check = triangolo(T);
  if(check){
    cout << "OK, " << e_accentata << " un triangolo" << endl;
  } else {
    cout << "non " << e_accentata << " un triangolo" << endl;
  }

// Tut 08.4. Stampa degli angoli del triangolo T
 titolo(" Tut 08.4.");
 stampa_angoli(T);

// Tut 08.5. Stampa dell'altezza di T
 double altezza = altezza_a(T);
 titolo(" Tut 08.5.");
 cout << "Altezza del triangolo = " << altezza << endl;

// Tut 08.6. Triangolo con una nuova altezza
 titolo(" Tut 08.6.");
 cout << "Inserire nuova altezza triangolo (es. 20)" << endl;
 cin >> altezza;
    T = cambia_altezza_a(T,altezza);
    cout << "nuovo triangolo" << endl;
    cout << " lato a = " << T.lato_a << " lato b = " << T.lato_b
     << " lato c = " << T.lato_c << endl;
 system("Pause"); system("CLS");
}
```

Settimana 09 – Vettori
"Linee di dati"

1. **Lezione 17: Vettori in C++ (Cap. 10.1-10.8 libro testo)**
2. **Lezione 18: Vettori in C++ (cap. 10.9 libro testo)**
3. **Tutorato 09: Strutture (ripasso)**

Il vettore dei primi dieci valori della successione di Fibonacci.

Turku, Finlandia, opera di Mario Merz del 1994, dal titolo:

"Fibonacci Sequence 1-55"

Lezione 17 - Vettori
Parte 1. Esempi (Cap. 10.1-10.8 libro testo)

In questa lezione introdurremo il vettori C++ e un ciclo, il ciclo **for**, alternativo ma equivalente al ciclo **while**.

Un vettore C++ è una sequenza di elementi tutti dello stesso tipo di lunghezza finita. *Insieme a un vettore dobbiamo sempre ricordarci di definire la sua lunghezza.* Come le stringhe, i vettori C++ costituiscono ***classe*** e richiedono una nuova libreria: <vector>. Un vettore V di tipo T si definisce in C++ con

vector<T> V;

oppure con:

vector<T> V(10,0); oppure **vector**<T> V(10,0.0);

Nel primo caso definiamo un vettore di lunghezza 0: la lunghezza, quando non precisata, vale 0. Nel secondo definiamo un vettore di 10 elementi tutti uguali a 0 (intero) oppure a 0.0 (reale). V ha indici che vanno da 0 a (n-1), la stessa convenzione vista per le stringhe.

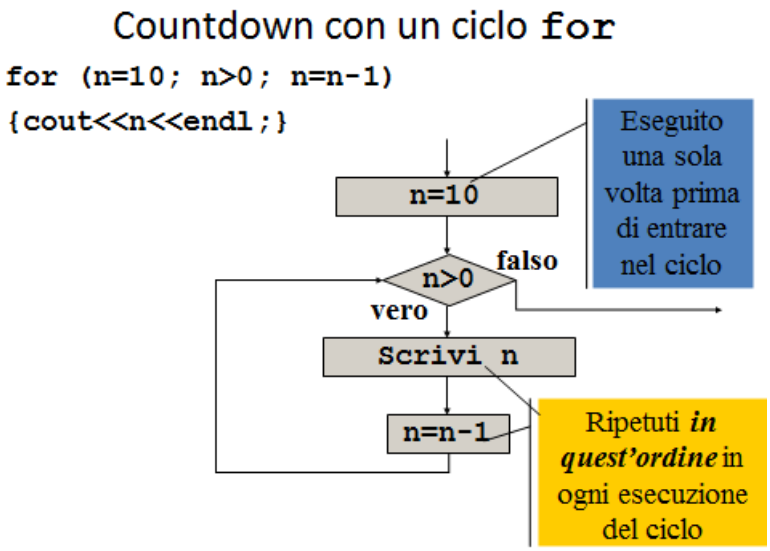
Se v è un vettore allora v[i] è il suo elemento numero i. Ogni v[i] puo' venire assegnato come se fosse una variabile, con v[i] = … . Le funzioni sui vettori date insieme alla classe dei vettori si scrivono in notazione ***prefissa***, come per le stringhe. Per esempio, la lunghezza n di un vettore V si calcola con

n = V.size(); (occhio a non scrivere n=V.size;)

Vediamo anche un ciclo equivalente al ciclo **while**, il ciclo **for**, che si scrive come segue (Cap. 10.3):

for (INITIALIZER; CONDITION; INCREMENTOR) {BODY;}

Il **for** esegue INITIALIAZER, poi finché CONDITION è vera esegue prima BODY e poi INCREMENTOR, se e quando CONDITION è falsa il **for** termina. Vediamo il funzionamento di un **for** su un esempio: il conto alla rovescia da 10 a 0, già visto nella Lezione 9 per spiegare il ciclo **while**.



Dunque un ciclo **for** in C++ equivale al seguente ciclo **while**:

INITIALIZER; while (CONDITION) {BODY;INCREMENTOR;}

Viceversa, ogni ciclo while si può scrivere anche come ciclo **for** (Cap. 10.4). All'interno di INITIALIZER, INCREMENTOR le diverse istruzioni si separano con una virgola, non un punto e virgola. Per esempio scriveremo:

```
for(i=0,j=10/*INIZIAL.*/; i<j /*COND.*/ ; i=i+1,j=j-1 /*INCREM.*/)
 { cout << "i = " << i << " j = " << j << endl; /*BODY*/ }
```

Questo è l'elenco delle funzioni sui vettori che vedremo oggi.

```
// Lezione17-Vettori. Sezioni 10.1-10.8
// Tipo dei vettori, vettore casuale, conto degli zeri
#include <iostream>
#include <math.h>
// Dobbiamo includere una nuova libreria per i vettori C++
#include <vector>
#include <stdlib.h>
using namespace std;

/* Cap. 10.8. Una funzione che stampa di un vettore */
void printVector (vector<int> V);

/* Cap. 10.8. Una funzione che genera un vettore casuale di n elementi */
vector<int> randomVector (int n, int upperBound);
```

```
/* Cap. 10.9. Una funzione che conta quanti elementi uguali al valore "value" ci sono in un vettore */
int howMany (vector<int> V, int value);


int main()
{int n,c,i,len,max;

 /* Cap.10.5. Ciclo che inserisce in un vettore dei valori letti da tastiera */
 vector<int> values (0,0); /* crea un vettore values di nome: values, di 0 elementi, tutti di valore 0 */
 cout << "Cap 10.5. Inserire un numero qualunque di numeri naturali" << endl; cout << "-1 per finire" << endl;
 cin>>c;
 while(c != -1)
 {
  values.push_back(c); /* incrementa di 1 la lunghezza del vettore quindi aggiunge il valore c nella nuova posizione creata */
  cin >> c;
 }
/* alla fine del while, il vettore "values" ha come lunghezza il numero degli elementi aggiunti */
 len=values.size(); // .size() calcola la lunghezza di "values"
 cout << "Stampa elementi inseriti" << endl;
 printVector(values);
 system("pause");system("CLS");

 /* Introduciamo il ciclo FOR. */
 cout << "Stampa di 4 valori casuali" << endl;
 for (i = 0; i < 4; i++)
  {int x = rand ();cout << x << endl;}
 system("pause");system("CLS");

 /* Cap. 10.8. Definiamo e sperimentiamo la funzione che genera un vettore casuale */
 n=10;max=1000;
 cout << "Cap. 10.8. Vettore casuale di lunghezza " << n << endl
  <<  "con valori da 0 fino a " << max-1 << endl << endl;
 printVector(randomVector(n,max));
 cout << endl;
 system("pause");system("CLS");


 /* Cap. 10.9. Sperimentiamo la funzione che conta un valore */
```

```
 vector<int> V (10,0); //vettore V con 10 volte zero
 cout << "Cap. 10.9. Stampa di un vettore V" << endl;
 printVector(V);
 cout << "Quanti zero in V? " << howMany(V,0) << endl;
 system("pause");system("CLS");}


/* DEFINIZIONE DI TUTTE LE FUNZIONI USATE NELLA LEZIONE */

/* Cap.10.8. Stampa di un vettore. */
void printVector (vector<int> V)
{int i, len = V.size(); // V.size() calcola la lunghezza di V
 for (i = 0; i<len; i++)
  {cout << V[i] << " ";}
 cout << endl;}

/* Cap.10.8. La funzione che genera un vettore casuale di n
elementi */
vector<int> randomVector (int n, int upperBound)
{vector<int> V (n);
 int i, len=V.size();
 for (i = 0; i<len; i++)
  {V[i] = rand () % upperBound;
   /* valore casuale da 0 a upperBound-1*/ }
 return V; //ricordatevi di restituire il valore
}

/* Cap. 10.9. la funzione che conta quanti elementi uguali al
valore value ci sono in un vettore */
int howMany (vector<int> V, int value)
{int i, len=V.size(); int count=0;
 for (i = 0; i<len; i++)
  {if (V[i] == value)
    {count++;}}
 return count; //ricordatevi di restituire il valore
}
```

Lezione 17 - Vettori
Parte 2. Esercizi

- Negli esercizi di questa lezione, e in generale in tutti gli esercizi sui vettori, si richiede di utilizzare un ciclo **for**, per prendere confidenza con questo tipo di cicli. Modificate i **for** visti a lezione.
- **Attenzione:** un vettore deve sempre avere una dimensione nota. Quindi scrivete **vector<double>** V(n,0.0); solo quando n è noto, e evitate di scrivere **int** n; **vector<double>** V(n,0.0); Non ottenete un vettore di lunghezza n "generica", come potreste aspettarvi. Ottenete un vettore con lunghezza n ***scelta a caso*** dal compilatore, anche negativa e quindi inaccettabile, oppure troppo grande. Un programma con queste istruzioni a volte funziona e a volta si blocca, non è prevedibile.
- Ricordatevi che **vector<T>** V; dà un vettore di 0 elementi.
- Per sommare, moltiplicare eccetera un vettore, prendete spunto dai cicli **while** che sommano o moltiplicano una successione, ma trasformateli in cicli **for**.
- Ricordate che le istruzioni all'interno dell'inizializzazione e dell'incremento di un **for** si separano con una virgola:

  for(**i=0,s=0;** i<len; ++i) {…}

  Invece il punto e virgola separa le tre parti del **for**.
- Tutti gli esercizi di questa lezione riguardano ***vettori di reali***, dunque di tipo ***vector<double>***.
- Per stampare un vettore di reali usate la funzione:

```
void printVett(vector<double> V)
 {int i, len=V.size();
  for(i=0;i<len;i=i+1)
   {cout << V[i] << " ";};
  cout << endl;}
```

- Usate l'esercizio 17.1 per costruire vettori da usare come esempi per gli altri esercizi.

Es 17.1 Definite una funzione

vector<double> primiinteri(int n)

che prende un intero n≥0 e restituisce il vettore di n reali di elementi: 1, …, n. Non usate un vettore di interi: ricordatevi che il C++ distingue tra un intero e un reale con lo stesso valore.

Es 17.2 Definite una funzione

double somma(vector<double> V)

che prende un vettore V = {x_0, …, x_{n-1}} di reali e restituisce la somma (x_0 + … + x_{n-1}) dei suoi elementi.

Es 17.3 Definite una funzione

double prodotto(vector<double> V)

che prende un vettore V = {x_0, …, x_{n-1}} di reali e restituisce il prodotto (x_0 * … * x_{n-1}) dei suoi elementi.

Es 17.4 Definite una funzione

double norma(vector<double> V)

che prende un vettore V = {x_0, …, x_{n-1}} di reali e ne restituisce la norma, definita come $\sqrt{(x_0^2 + \ldots + x_{n-1}^2)}$.

Es 17.5 Definite una funzione

double prodottoscalare(vector<double> V, vector<double> W)

che prende due vettori V = {x_0, …, x_{n-1}}, W = {y_0, …, y_{n-1}} di reali di uguale lunghezza e ne restituisce il prodotto scalare, definito come (x_0*y_0 + … + x_{n-1}*y_{n-1}).

Es 17.6 Definite una funzione

vector<double> prodottoperscalare(vector<double> V, double x)

che prende un vettore di reali V = {x_0, …, x_{n-1}}, un numero reale x, e restituisce il vettore V*x = {x_0*x, …, x_{n-1}*x} (con il prodotto per componenti).

Es 17.7 Definite una funzione

vector<double> sommavettoriale(vector<double> V, vector<double> W)

che prende due vettori di reali V = {x_0, …, x_{n-1}}, W = {y_0, …, y_{n-1}} di uguale lunghezza e ne restituisce la somma vettoriale, definita come V+W = {x_0+y_0, …, x_{n-1}+y_{n-1}} (il vettore ottenuto sommandoli elemento per elemento).

Lezione 17 - Vettori
Parte 3. Soluzioni

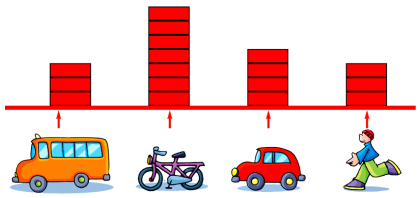
```
// Lezione 17-Vettori. Soluzioni degli esercizi

#include <iostream>
#include <math.h>
#include <vector>
#include <stdlib.h>
using namespace std;

void printVett(vector<double> V)
 {int i, len=V.size();
  for(i=0;i<len;i=i+1){cout << V[i] << " ";}
  cout << endl;
 }

/* Es 17.1 vettore dei primi n interi positivi */
vector<double> primiinteri(int n)
{vector<double> V (n, 0);
 int i;
 for(i=0;i<n;i=i+1) {V[i]=i+1;}
 return V; /* ricordatevi di restituire il valore */ }

/* Es 17.2 somma di un vettore di reali */
double somma(vector<double> V)
{int i, len=V.size(); double s;
 for(i=0, s=0;i<len;i=i+1) //separiamo i=0 da s=0 con una virgola
  {s=s+V[i];}
 return s;}

/* Es 17.3 prodotto di un vettore di reali */
double prodotto(vector<double> V)
{int i, len=V.size(); double s;
 for(i=0, s=1;i<len;i=i+1) //separiamo i-0 da s=1 con una virgola
  {s=s*V[i];}
 return s;}

/* Es 17.4 norma di un vettore di reali */
double norma(vector<double> V)
{int i, len=V.size(); double s;
 for(i=0, s=0;i<len;i=i+1) //separiamo i-0 da s=0 con una virgola
  {s=s+V[i]*V[i];}
 return sqrt(s);}
```

```
/* Es 17.5 prodotto scalare di vettori reali */
double prodottoscalare(vector<double> V, vector<double> W)
//supponiamo i due vettori della stessa lunghezza
{int i, len=V.size(); double s;
 for(i=0, s=0;i<len;i=i+1) //separiamo i-0 da s=0 con una virgola
  {s=s+V[i]*W[i];}
 return s;}

/* Es 17.6 prodotto per scalare di vettori reali */
vector<double> prodottoperscalare(vector<double> V, double x)
{int i, len=V.size(); vector<double> W (len,0);
 for(i=0;i<len;i=i+1)
  {W[i]=V[i]*x;}
 return W; /* ricordatevi di restituire il valore */ }

/* Es 17.7 somma vettoriale di vettori reali */
vector<double> sommavettoriale(vector<double> V, vector<double> W)
// supponiamo V, W della stessa lunghezza
{int i, len=V.size(); vector<double> Z (len,0);
 for(i=0;i<len;i=i+1)
  {Z[i]=V[i]+W[i];}
 return Z; /* ricordatevi di restituire il valore */ }

int main(){int n = 7;
 cout << "Es 17.1 vettore dei primi " << n <<
  " interi positivi" << endl;
 printVett(primiinteri(n)); cout << endl;
 system("pause");system("CLS");

 cout << "Es 17.2 somma vettore dei primi " << n <<
  " interi positivi" << endl;
 cout << "somma(primiinteri(n))) = " <<
  somma(primiinteri(n)) << endl;
 system("pause");system("CLS");

 cout << "Es 17.3 prodotto vettore dei primi " << n
  <<" interi positivi" << endl;
 cout << "prodotto(primiinteri(n)) = " <<
  prodotto(primiinteri(n)) << endl;
 system("pause");system("CLS");

 cout << "Es 17.4 norma vettore dei primi " <<
  n <<" interi positivi" << endl;
```

```
 cout << "norma(primiinteri(n)) = " <<
  norma(primiinteri(n)) << endl;
 system("pause");system("CLS");

 cout << "Es 17.5 prodotto scalare vettore dei primi " << n <<"
interi per se stesso" << endl;
 cout << "prodottoscalare(primiinteri(n),primiinteri(n)) = " <<
  prodottoscalare(primiinteri(n),primiinteri(n)) << endl;
 system("pause");system("CLS");

 cout << "Es 17.6 prodotto per " << n << " del vettore dei primi "
  << n << " interi" << endl;
 cout << "prodottoperscalare(primiinteri(n),n) = "
  << endl;
 printVett(prodottoperscalare(primiinteri(n),n));
  cout << endl;
 system("pause");system("CLS");

 cout << "Es 17.6 somma vettoriale vettore dei primi " << n <<
  "interi" << endl;
 cout << "sommavettoriale(primiinteri(n),primiinteri(n)) = "
  << endl;
 printVett(sommavettoriale(primiinteri(n),primiinteri(n)));
  cout << endl;
 system("pause");system("CLS");
}
```

Lezione 18 - Vettori
Parte 1. Esempi (Cap. 10.10)
Un "istogramma"

Un ***"vettore di frequenze"*** è un vettore che descrive il numero di volte con cui certi elementi compaiono in un elenco dato. Viene di solito rappresentato da un ***"istogramma"***, una serie di colonne la cui altezza rappresenta la frequenza. Per esempio, supponiamo che ci sono 17 bambini in una classe di scuola elementare, e che 3 vanno a scuola in autobus, 7 in bicicletta, 4 in auto e 3 a piedi. Allora il "vettore delle frequenze" dei mezzi di trasporto è {3,7,4,3} e questo è il suo istogramma:



In questa lezione vediamo un esempio tratto dal capitolo 10.10 del libro di testo, il calcolo del "vettore delle frequenze" di un vettore casuale. Generiamo un vettore casuale di 100 mila elementi di valori tra 0 e 9, e contiamo quanti 0, quanti 1, eccetera compaiono ne vettore: verifichiamo che sono circa un decimo.
Nella seconda ora vedremo diversi esercizi del livello di difficoltà degli esercizi di esame.

```
/* Lezione18-Vettori. Cap. 10.10
Conto degli elementi uguali a un valore dato in un vettore
casuale. */

#include <iostream>
#include <math.h>
#include <vector>
#include <stdlib.h>
using namespace std;

/* Innanzitutto includiamo tutte le funzioni viste nella lezione
precedente, ovvero: la funzione che stampa un vettore */
void printVector (vector<int> V)
{int i, len = V.size();
 for (i = 0; i<len; i++)
  {cout << V[i] << " ";}
```

```
 cout << endl;}

//la funzione che genera un vettore casuale di n elementi
vector<int> randomVector (int n, int upperBound)
{vector<int> V (n);
 int i, len=V.size();
 for (i = 0; i<len; i++)
  {V[i] = rand () % upperBound;}
 return V; //ricordatevi di restituire il valore
}

/* la funzione che conta quanti elementi uguali al valore value ci
sono in un vettore */
int howMany (vector<int> V, int value)
{int i, len=V.size(); int count=0;
 for (i = 0; i< len; i++)
  {if (V[i] == value) count++;}
 return count; //ricordatevi di restituire il valore
}

/* Ora generiamo un vettore casuale di n elementi di valore da 0 a
upperBound-1, e per ogni possibile valore contiamo quante volte
compare. Il risultato e' all'incirca (n/upperBound). Costruiremo
un vettore delle frequenze per i valori del vettore casuale. Con
un abuso di linguaggio, chiameremo il vettore delle frequenze
"istogramma" (questo nome andrebbe riservato al grafico).  */

int main()
{cout << "Generiamo un vettore di valori casuali" << endl;
 int numValues      = 100000;
 int upperBound     = 10;
/*  Questo vettore conterra' il vettore delle frequenze o
"istogramma". */
 vector<int> histogram (upperBound,0);
/* i=0, …, numValues-1, val=0, …, upperBound-1 */
 int i, val;
/*  Ora generiamo un vettore vector contenente numValues valori
casuali, scelti tra 0 e upperBound-1 */
 vector<int> vector = randomVector (numValues, upperBound);
/* Ci aspettiamo che ogni valore compaia un numero di volte pari
a: (numValues/upperBound)*/
 double expected    = numValues/((double) upperBound); /* dobbiamo
convertire a reale per evitare l'arrotondamento */
 cout << " valori = " << numValues << "\n massimo = "
```

```
  << (upperBound-1) << "\n elementi previsti per ogni valore = "
<< expected << endl;

/* SOLUZIONE 1. Usando la funzione HowMany, contiamo quanti
elementi troviamo per ogni valore e calcoliamo la differenza in
percentuale dal numero previsto */
 cout << endl << "Istogramma. Soluzione 1" << endl;
 cout << endl << "Valori\tQuanti\tDifferenza in %" << endl;
 for (i = 0; i<upperBound; i++)
  {int h = howMany (vector, i);
   double diff = 100*(expected-h)/expected;
   cout << i << "\t" << h << "\t" << diff << endl;};
 system("pause"); system("CLS");

/* SOLUZIONE 2. Il difetto della soluzione 1 è che utilizza la
funzione howMany tante volte quanti sono i valori da cercare. Ogni
volta howMany percorre l'intero vettore per contare quante volte
compare un dato valore: se ci sono 10 valori lo percorre 10 volte.
Ora includiamo una seconda soluzione, piu' efficiente, che
percorre il vettore una volta sola. */
 cout << "Istogramma. Soluzione 2." <<
  "\n Percorriamo il vettore una volta sola" << endl;
/* Usiamo un vettore istogram, che all'inizio contiene solo zeri.
Ogni volta che troviamo il valore val incrementiamo di 1
istogram[val]. Alla fine istogram[val] indica quante volte compare
il valore val nel vettore vector */
  cout << "\n valori = " << numValues << "\n massimo = " <<
  upperBound-1 << "\n elementi per valore = " << expected << endl;

for (i = 0; i<numValues; i++)
  {
   int val = vector[i]; //troviamo val nella posizione i di vector
   histogram[val]++;    //aggiorniamo quante volte compare val
  }
 /* Per ogni valore stampiamo il risultato, insieme alla
differenza in percentuale dal numero previsto */
 cout << endl << "Valori\tQuanti\tDifferenza in %" << endl;
 for (val = 0; val<upperBound; val++)
  {int h = histogram[val];
   double diff = 100*(expected-h)/expected;
   cout << val << "\t" << h << "\t" << diff << endl;}
 system("pause");system("CLS");}
```

Lezione 18 - Vettori
Parte 2. Esercizi

- Negli esercizi di questa lezione, e in generale in tutti gli esercizi sui vettori, si consiglia di utilizzare un ciclo **for**, per prendere confidenza con questo tipo di cicli. Modificate i cicli **for** visti a lezione.
- Ricordate che le istruzioni all'interno dell'inizializzazione e dell'incremento di un **for** si separano con una virgola:

  for(**i=0,s=0;** i<len; ++i) {…}
- Tutti gli esercizi di questa lezione riguardano ***vettori di interi***, dunque di tipo ***vector<int>***.
- Per stampare un vettore di interi usate la funzione:

```
void printVett(vector<int> V){int i, len=V.size();
for(i=0;i<len;i=i+1){cout << V[i] << " ";}; cout << endl;}
```

- Tutti gli esempi proposti per controllare gli esercizi riguardano i vettori v = (1,2,3,4,5,6,7,8,9) e w = (1,2,3,4,5,7,6,8,9,0). Dovrete costruirli a partire da una dichiarazione ***vector<int> v(9,0), w(10,0)*** e assegnazioni.
- Prestate particolare attenzione agli esercizi 18.1, 18.2, 18.3. In 18.1, 18.2 spieghiamo come decidere se ***tutti gli elementi di un vettore*** soddisfano una certa proprietà. In 18.3 spieghiamo come decidere se ***almeno un elemento di un vettore*** soddisfa una certa proprietà. Avevamo già esaminato questi problemi nelle soluzioni degli esercizi 13.2 e 14.2.

Es. 18.1. Scrivete una funzione booleana

bool tutti_positivi(**vector**<**int**> v)

che prende un vettore di interi e restituisce 1 (true) se tutti gli elementi di v sono positivi, e 0 (false) altrimenti. **Suggerimento.** Usate un ciclo per controllare se v[i]>0 per tutti gli i=0, …, (lunghezza di v)-1. Non appena incontrate un i tale che v[i]<=0 terminate la funzione e restituite **false.** Se uscite dal ciclo dopo aver esplorato l'intero vettore (se quindi tutti gli elementi di v soddisfano v[i]>0) restituite invece **true.** **Esempi.** Con v, w come sopra: tutti_positivi(v) = **true,** mentre tutti_positivi(w) = **false.**

Es. 18.2. Scrivete una funzione booleana

bool ordinato(**vector**<**int**> v)

che prende un vettore di interi e restituisce 1 (true) se v è ordinato in modo debolmente crescente, e 0 (false) altrimenti.

Suggerimento. Usate un ciclo per controllare se v[i]<=v[i+1] per tutti gli i=0, …, (lunghezza di v)-2. Non appena incontrate un i tale che v[i]>v[i+1] terminate la funzione restituite **false.** Se il ciclo finisce (se quindi tutti gli elementi di v soddisfano v[i]<=v[i+1]) restituite invece **true. Esempi.** Con v, w come sopra: ordinato(v) = **true**, ordinato(w) = **false.**

Es 18.3. Scrivere una funzione

bool appartiene(**int** x, **vector<int> v**)

che decida se un intero x appartiene o no a un vettore v di interi. **Suggerimento.** Usate un ciclo per controllare se v[i]==x per tutti gli i=0, …, n-1. Non appena incontrate un i tale che v[i]==x restituite **true.** Se uscite dal ciclo dopo aver esplorato l'intero vettore (se quindi non avete trovato x in v) restituite invece **false. Esempi.** Con v, w come sopra: appartiene(0,w) = appartiene(1,v) = appartiene(7,v) = appartiene(9,v) = **true**, mentre appartiene(0,v) = appartiene(10,v) = **false.**

Es. 18.4. Definite una funzione

int minimo(**vector<int>** v)

che restituisce il valore minimo valore del vettore v. **Suggerimento.** Usate una variabile m per contenere il minimo. All'inizio m vale v[0]. Usate un ciclo per controllare se m<=v[i] per tutti gli i=1, …, n-1. Ogni volta che incontrate un i tale che m>v[i] rimpiazzate m con v[i]. Quando uscite dal **while** dentro m si trova il piu' piccolo valore visto. Restituite m. **Esempio.** Con v, w come sopra: minimo(v)=1, minimo(w)=0.

Es. 18.5. Definite una funzione

int indicedelminimo(**vector<int>** v)

che restituisce un qualsiasi indice del valore minimo di un vettore (possono essercene diversi). **Nota.** Attenzione alla differenza con l'esercizio precedente: l'indice i del minimo valore m di un vettore è un i tale che v[i]=m, non va confuso con il valore m del minimo. **Suggerimento.** Usate due variabili m, ind per contenere minimo e indice del minimo. All'inizio m, ind valgono v[0] e 0. Usate un ciclo per controllare se m<=v[i] per tutti gli i=1, …, n-1. Ogni volta che incontrate un i tale che m>v[i] rimpiazzate m con v[i] e ind con i. Quando uscite dal **while** dentro m si trova il piu' piccolo valore visto, e dentro ind il suo indice. Restituite ind. **Esempio.** Con v, w come sopra: indicedelminimo(v)=0, indicedelminimo(w)=9.

Lezione 18 - Vettori
Parte 3. Soluzioni

```
// Lezione18-Vettori. Esercizi

#include <iostream>
#include <math.h>
#include <vector>
#include <stdlib.h>
using namespace std;

void printVett(vector<int> v)
 {int i, len=v.size();
  for(i=0;i<len;i=i+1){cout << v[i] << " ";}
  cout << endl;}

/* Es. 18.1. controllo se tutti gli elementi di un vettore sono
positivi */
bool tutti_positivi(vector<int> V)
 {int i, len = V.size();
  for(i=0;i<len;i++)
   //se trovo un non-positivo la risposta e' falso
   {if (V[i] <= 0) return false;}

/* solo dopo aver controllato in tutto V che non ci sono non-
positivi posso affermare che tutti gli elementi di V sono positivi
*/
  return true;
 }

/* Es. 18.2. test di ordinamento di un vettore */
bool ordinato(vector<int> v)
{int i, len=v.size();
 for(i=0; i<len-1; i=i+1)
  {
   if (v[i]>v[i+1])
    {return false;}
  }
 return true;
};

/* Es 18.3. appartenenza a un vettore */
bool appartiene(int x, vector<int> v)
{int i, len=v.size();
 for(i=0; i<len; i=i+1)
```

```
  {
   if (v[i]==x)
    {return true;} //non appena trovo x restituisco i
  }
//se termino il ciclo senza trovare x, allora x non è in v
 return false;
};

/* Es. 18.4. minimo di un vettore */
int minimo(vector<int> v)
{int i, len=v.size(); int m;
 for(i=1, m=v[0]; i<len; i=i+1)
  {
   if (m>v[i])
    {m=v[i];}
  }
 return m; //ricordatevi di restituire il valore
};

/* Es. 18.5. Indice del minimo di un vettore */
int indicedelminimo(vector<int> v)
{ int i, len=v.size(); int m, ind;
 for(i=1, m=v[0], ind=0; i<len; i=i+1)
  {
   if (m>v[i])
    {m=v[i];ind=i;}
  }
 return ind; //ricordatevi di restituire il valore
};

int main(){vector <int> v(9,0), w(10,0);
/* definiamo v = (1,2,3,4,5,6,7,8,9) e w = (1,2,3,4,5,7,6,8,9,0).
Tutti gli esempi riguarderanno questi due vettori */
 int i; for(i=0;i<9; i=i+1) {v[i]=i+1;w[i]=i+1;};
/* Non ho bisogno di porre w[9]=0 perché w[9] vale già 0. */

 cout << "Es. 18.1. Controllo se tutti gli elementi di un vettore
sono positivi" << endl;
 cout << "vettore v: "; printVett(v);
 cout << "vettore w: "; printVett(w);
 cout << "tutti_positivi(v) = " << tutti_positivi(v) << endl;
 cout << " tutti_positivi(w) = " << tutti_positivi(w) << endl;
 system("pause");system("CLS");

 cout << "Es. 18.2. Test di ordinamento di un vettore" << endl;
```

```
 cout << "vettore v: "; printVett(v);
 cout << "vettore w: "; printVett(w);
 cout << "ordinato(v) = " << ordinato(v) << endl;
 cout << "ordinato(w) = " << ordinato(w) << endl;
 system("pause");system("CLS");

 cout << "Es 18.3. Appartenenza a un vettore" << endl;
 cout << "vettore v: "; printVett(v);
 cout << "vettore w: "; printVett(w);
 cout << "appartiene(0,w)  = " << appartiene(0,w) << endl;
 cout << "appartiene(1,v)  = " << appartiene(1,v) << endl;
 cout << "appartiene(7,v)  = " << appartiene(7,v) << endl;
 cout << "appartiene(9,v)  = " << appartiene(9,v) << endl;
 cout << "appartiene(0,v)  = " << appartiene(0,v) << endl;
 cout << "appartiene(10,v) = " << appartiene(10,v) << endl;
 system("pause");system("CLS");

 cout << "Es. 18.4. Minimo di un vettore " << endl;
 cout << "vettore v: "; printVett(v);
 cout << "vettore w: "; printVett(w);
 cout << "minimo(v)  = " << minimo(v) << endl;
 cout << "minimo(w)  = " << minimo(w) << endl;
 system("pause");system("CLS");

 cout << "Es. 18.5. Indice del minimo di un vettore" << endl;
 cout << "vettore v: "; printVett(v);
 cout << "vettore w: "; printVett(w);
 cout << "indicedelminimo(v)  = " << indicedelminimo(v) << endl;
 cout << "indicedelminimo(w)  = " << indicedelminimo(w) << endl;
 system("pause");system("CLS");

}
```

Tutorato 09 – Strutture (ripasso)
Parte 1. Esercizi proposti

La struttura Passeggiata. Definire la struttura Passeggiata a partire da due dati: la velocità, rappresentata in m/sec (un numero reale), e la durata, in ore, minuti e secondi, rappresentata dalla struttura vista nella Lezione 16:

```
struct Time {int hour, minute; double second;};
```

Supporremo che tutte le passeggiate si svolgano lungo il perimetro di un triangolo T, percorrendo nell'ordine i lati a, b, c anche più volte. T viene fornito a parte. È possibile conoscere l'esatta posizione alla fine della passeggiata di un oggetto, poiché quest'ultimo si muove lungo un percorso noto ad una velocità costante. Per rappresentare T, riutilizzate la struttura che rappresenta i lati di un triangolo vista nel Tutorato 08:

```
struct Triangolo {double lato_a, lato_b, lato_c;};
```

Riutilizzate le funzioni viste nel Tutorato 08 che: stampano un triangolo (Tut 08.2), controllano se tre lati definiscono un triangolo (Tut 08.3).

Tut 09.1 Scrivere una funzione

```
void stampaPasseggiata(Passeggiata p);
```

che prende una passeggiata p e ne stampa i dati relativi a velocità e durata.

Tut 09.2 Scrivere una funzione

```
void inserisciPasseggiata(Passeggiata& p);
```

che prende una variabile p di tipo Passeggiata e richiede all'utente di inserire i dati da tastiera, quindi li stampa per verifica, usando la funzione precedente. Il passaggio per riferimento della variabile p, indicato dal simbolo &, è necessario per modificare il contenuto della variabile p originaria e non agire su una copia.

Tut 09.3 Definire una funzione

```
void stampaGiriCompleti(Passeggiata p, Triangolo T)
```

che prende una passeggiata P lungo il perimetro di un triangolo T definito come nell'esercizio precedente, e stampa il numero di giri completi compiuti al termine della passeggiata.

Tut 09.4. Definire una funzione

```
void percentuale_ultimo_giro(Passeggiata P, Triangolo T);
```

che prendendo come parametro il percorso rappresentato tramite un triangolo definito come nell'esercizio precedente, stampi la

percentuale dell’ultimo giro non ancora completato al termine della passeggiata.

Esempi. Verificare che Luigi dopo una passeggiata di 11 minuti ad una velocità di 0.3 m/s lungo il perimetro di una aiuola triangolare di lati 3,4,5 abbia compiuto 16 giri completi. Inoltre con gli stessi dati verificare che la percentuale dell’ultimo giro non ancora completato sia del 50%.

Tutorato 09 – Strutture (ripasso)
Parte 2. Soluzioni

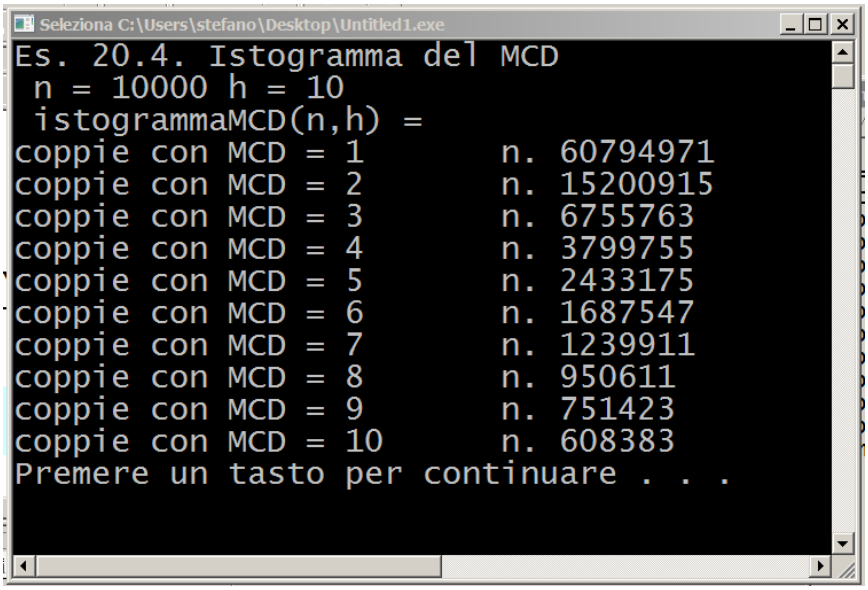
```
// Tutorato 09 - Soluzioni
#include <iostream>
#include <string>
#include <stdlib.h>
#include <math.h>
using namespace std;

/* non dimenticate il punto e virgola dopo ogni definizione di
struttura */
struct Triangolo {double lato_a, lato_b, lato_c;};
struct Time{int hour, minute; double second;};
struct Passeggiata{double velocita;Time durata; };

// Riutilizzo delle funzioni sui triangoli
// Funzione per inserire i dati relativi al triangolo T
// copiata dall'esercizio Tut 08.2
void inserisciTriangolo(Triangolo& T)
//Uso il passaggio per riferimento per poter modificare T
{
 cout << "Inserire i 3 lati del triangolo" << endl;
 cout << "lato a (la base del triangolo)" << endl;
 cin >> T.lato_a;
 cout << "lato b" << endl;
 cin >> T.lato_b;
 cout << "lato c" << endl;
 cin >> T.lato_c;
 cout << "Inseriti i lati \n a = " << T.lato_a << " (la base)"
  << "\n b  = " << T.lato_b << "\n c = " << T.lato_c << endl;
}

// Funzione per controllare se T e' un triangolo
// Copiata da Tut08.3
bool triangolo (Triangolo T)
{double a = T.lato_a, b = T.lato_b, c = T.lato_c;
 bool condizione_per_triangolo =
  (a < b+c ) && (b < a+c ) && (c < a+b);
 return condizione_per_triangolo;
}

//Tut 09.1 Funzione che stampa i dati relativi alla passeggiata p
void stampaPasseggiata(Passeggiata p)
```

```
{cout<<p.durata.hour<<":"<<p.durata.minute<<":"
  <<p.durata.second<<endl;
 cout<<"Velocita` "<<p.velocita<<endl;}


//Tut09.2 funzione che chiede i dati relativi alla passeggiata p
void inserisciPasseggiata(Passeggiata& p)
//Uso il passaggio per riferimento per modificare p
{
 cout<<"Inserire le ore di passeggiata: ";
 cin>> p.durata.hour;
 cout<<"Inserire i minuti di passeggiata: ";
 cin>>p.durata.minute;
 cout<<"Inserire i secondi di passeggiata: ";
 cin>>p.durata.second;
 cout<<"Inserire la velocita` della passeggiata (m/s): ";
 cin>>p.velocita;
}

//soluzione Tut09.3
void stampaGiriCompleti(Passeggiata p, Triangolo T)
{
double perimetro=T.lato_a+T.lato_b+T.lato_c;
int secondi=p.durata.second+p.durata.minute*60+p.durata.hour*3600;
double distanzaPercorsa=secondi*p.velocita;
int giri=distanzaPercorsa/perimetro;
cout<<"Giri completi: "<<giri<<endl;
}

//soluzione Tut09.4
void stampaPercentualeUltimoGiro(Passeggiata p,Triangolo percorso)
{
double perimetro=percorso.lato_a+percorso.lato_b+percorso.lato_c;
int secondi=p.durata.second+p.durata.minute*60+p.durata.hour*3600;
double distanzaPercorsa=secondi*p.velocita;
int ultimoGiro=(int)distanzaPercorsa%(int)perimetro;
cout<<"Percentuale dell'ultimo giro completata: "
 << ultimoGiro/perimetro*100<<"%"<<endl;
}

//funzione utilizzata per stampare un titolo (non richiesta)
void titolo(string s)
{string riga = "-----------------------------";
 cout << riga << endl << s << endl << riga << endl; }
```

```
int main() {
 char e_accentata = '\212'; //codice ASCII per "e accentata"
 Triangolo T;

 titolo(" Tut 10. La struttura Passeggiata");
 Passeggiata p; inserisciPasseggiata(p);
// inserimento dei lati del percorso triangolare T
 inserisciTriangolo(T);
 if (triangolo(T)==false)
  {cout<<"Il percorso non "<<e_accentata<<" triangolare"<<endl;
   system("pause"); exit(1);}
   //in questo caso termina il programma

// Tut 09.5.
 titolo(" Tut 10.03.");
 stampaGiriCompleti(p,T); cout<<endl;
// Stampa del titolo
 titolo(" Tut 10.04.");
 stampaPercentualeUltimoGiro(p,T); cout<<endl;

 titolo(" Controllo l'esempio richiesto per la passeggiata ");
 cout << " 11 minuti a 0.3 m/s attorno a un triangolo di lati
3,4,5." << endl;
 T={3,4,5}; Time d={0,11,0}; p={0.3,d};
 stampaPasseggiata(p);
 stampaGiriCompleti(p,T);
 stampaPercentualeUltimoGiro(p,T);

 cout<<endl; system("pause");
}
```

Settimana 10
Esercizi sui Vettori

1. **Lezione 19: Esercizi sui vettori**
2. **Lezione 20: Esercizi su vettori**
3. **Tutorato 10: Vettori.**



Usando un vettore delle frequenze, possiamo verificare empiricamente che la probabilità che due numeri siano primi tra loro vale $6/\pi^2 = 0{,}6079 \ldots$*, come accennato nella Presentazione.*

Lezione 19: esercizi sui vettori
Parte 1. Esercizi

- Si consiglia di utilizzare un ciclo **for**.
- Tutti gli esercizi di questa lezione riguardano ***vettori di interi***, dunque di tipo ***vector<int>***.
- Per stampare un vettore di interi usate la funzione:

```
void printVett(vector<int> V){int i, len=V.size();
for(i=0;i<len;i=i+1){cout << V[i] << " ";}; cout << endl;}
```

- In tutti gli esercizi di questa lezione utilizziamo come esempi i seguenti vettori di interi:
  V=(1,2,1), W=(1,2,2,1), Z=(1,2,2), T=(1,2,3,-3,2,1)
  e le proprietà p, q definite da

```
bool p(int x) {return (x<0);}
bool q(int x) {return (x>2);}
bool r(int x) {return (x>0);}
```

  p rappresenta la proprietà di essere negativi, q la proprietà di avere valore>2, ed r la la proprietà di essere positivi. Attenzione, p,q,r sono funzioni e quindi *vanno definite fuori dal main.*

Es. 19.1 ***(Controllo se due vettori sono uguali elemento per elemento)***. Scrivete una funzione

bool uguali(**vector<int>** v, **vector<int>** w)

che prenda come argomenti due vettori v, w di interi, e restituisca: true se i due vettori sono uguali (se hanno la stessa lunghezza e contengono gli stessi elementi nello stesso ordine), e false se non lo sono. **Suggerimento.** Calcolate la lunghezza len di v, e len2 di w: se sono diverse restituite false, altrimenti continuate. Usate un ciclo per controllare se v[i]==w[i] per i=0, …, len-1. Non appena trovate due elementi diversi, restituite false. Se arrivate alla fine del ciclo, allora non ci sono elementi di v, w nella stessa posizione e diversi: in questo caso restituite true. **Esempi.** Se V, W, Z, T sono come sopra, allora uguali(V,V) = 1, uguali(V,Z) = uguali(W,Z) = 0.

Es 19.2. Scrivere una funzione

bool simmetrico(**vector<int>** v)

che prenda come argomenti un vettore v, e restituisca true se il vettore è simmetrico, cioe' se due valori in posizione equidistante dal centro sono uguali. Per esempio V=(1,2,1) e

W=(1,2,2,1) sono simmetrici, mentre Z=(1,2,**2**), T=(1,2,3,**-3**,2,1) non lo sono. **Suggerimento.** Calcolate la lunghezza lun di v, poi usate un ciclo per controllare se v[i]==v[j], per i=0 e j=len-1, quindi per i=1 e j=len-2, eccetera. Continuate a controllare finche' (i<j). Non appena trovate due valori diversi restituite false. Se uscite dal ciclo, allora due qualunque valori in posizioni equidistanti dal centro sono uguali. In questo caso restituite true. **Esempi.** Se V, W, Z, T sono come sopra, allora simmetrico(V)=simmetrico(W)=1, simmetrico(Z)=simmetrico(T)=0.

Es 19.3. Scrivere una funzione

void inverti(**vector<int>&** v)

che prende un vettore v e lo modifica, disponendo i suoi elementi in ordine inverso. E' necessario usare il passaggio per riferimento, indicata dal simbolo &. **Suggerimento.** Calcolate la lunghezza lun di v, poi usate un ciclo per scambiare i valori di v[i], v[j], per i=0 e j=len-1, quindi per i=1 e j=len-2, eccetera. Continuate a scambiare finche' (i<j). A questo punto smettete, *altrimenti rimettete gli elementi al loro posto*. Ricordatevi che per scambiare v[i], v[j] dovete prima salvare v[i] in una variabile temporanea. **Esempi.** Se V, W, Z, T sono come sopra, e applico inverti(V), inverti(W) allora V, W essendo simmetrici non cambiano. Se applico inverti(Z), inverti(Z) allora Z diventa (2,2,1) e T diventa (1,2,-3,3,2,1).

Es 19.4. Scrivere una funzione

bool esiste(**vector**<**int**> v, **bool** p(**int** x))

che dato un vettore v e una funzione **bool** p(**int** x) restituisca true se esiste un indice i in v tale che p(v[i])=true, e restituisca false se non ci sono i in v tali che p(v[i])=true. **Suggerimento.** Calcolate la lunghezza len di v, poi usate un ciclo per calcolare p(v[i]) per i che varia da 0 a len-1. Non appena trovate un i tale che p(v[i])==true restituite true. Se terminate il ciclo, vuol dire che non ci sono indici i in v tali che p(v[i])==true. In tal caso restituite false. **Esempi.** Siano p, q le proprietà di essere negativi e di essere > 2 definite sopra. Allora esiste(V,p) = esiste(W,p)= esiste(Z,p) = **false** perche' V, W, Z non contengono valori negativi. Invece esiste(T,p) = **true**, perche' T contiene -3. Per un ragionamento simile abbiamo: esiste(V,q) = esiste(W,q) = esiste(Z,q) = **false** e esiste(T,q) = **true**.

Es 19.5. Scrivere una funzione

int conta(**vector**<**int**> v, **bool** p(**int** x))

che dato un vettore v e una funzione **bool** p(**int** x) restituisca il numero di indici i in v tale che p(v[i])=true (dunque 0 se non ci sono i in v tali che p(v[i])=true). **Suggerimento.** Calcolate la lunghezza len di v, poi usate un ciclo per calcolare p(v[i]) per i che varia da 0 a len-1 e un contatore count inizializzato a 0. Ogni volta che trovate un i tale che p(v[i])==true incrementate count di 1. Quando terminate il ciclo, restituite count. **Esempi.** Siano p, r le proprietà di essere negativi e di essere positivi

definite sopra. Allora conta(V,p) = conta(W,p)= conta(Z,p) = 0 perche' V, W, Z non contengono valori negativi. Invece conta(T,p) = 1, perche' T contiene un valore negativo, -3. Invece conta(V,r) = 3, conta(W,r) = 4, conta(Z,r) = 3 e conta(T,r) = 5.

Es 19.6. Scrivere una funzione

```
int primoindice(vector<int> v, bool p(int x) )
```

che dato un vettore v e una funzione **bool** p(**int** x) restituisca il primo indice i in v tale che p(v[i])=true, e restituisca -1 se non ci sono i in v tali che p(v[i])=true. **Suggerimento.** Calcolate la lunghezza len di v, poi usate un ciclo per calcolare p(v[i]) per i che varia da 0 a len-1. Non appena trovate un i tale che p(v[i])==true restituite i. Se terminate il ciclo, vuol dire che non ci sono indici i in v tali che p(v[i])==true. In tal caso restituite -1. **Esempi.** Siano p, q le proprietà di essere negativi e di essere > 2 definite sopra. Allora primoindice(V,p) = primoindice(W,p)= primoindice(Z,p) = -1 perche' V, W, Z non contengono valori negativi. Invece primoindice(T,p) = 3, perche' 3 è la posizione di -3, il primo valore negativo di T. Per un ragionamento simile abbiamo: primoindice(V,q) = primoindice(W,q) = primoindice(Z,q) = -1 e primoindice(T,q) = 2.

Lezione 19: esercizi sui vettori
Parte 2. Soluzioni

```
// Lezione19-Esercizi sui vettori.
#include <iostream>
#include <math.h>
#include <vector>
#include <stdlib.h>
using namespace std;

void printVett(vector<int> V)
 {int i, L=V.size();
  for(i=0;i<L;i=i+1){cout << V[i] << " ";}
  cout << endl;}

void printVett(vector<int> v, vector<int> w,
 vector<int> z, vector<int> t)
{cout << "vettore V: "; printVett(v);
 cout << "vettore W: "; printVett(w);
 cout << "vettore Z: "; printVett(z);
 cout << "vettore T: "; printVett(t);}

/* Es. 19.1 Test di uguaglianza di vettori */
bool uguali(vector<int> v, vector<int> w)
{
 int i, len = v.size(), len2 = w.size();
/*se v,w hanno lunghezze diverse sono diversi */
 if (len!=len2) {return false;}
/*altrimenti confronto v, w elemento per elemento */
 for(i=0;i<len;i=i+1)
  {if (v[i]!=w[i]) {return false;}}
/*se il ciclo finisce, allora non ho trovato due valori di v, w
nella stessa posizione e diversi*/
 return true;
}

/* Es 19.2. Test di simmetria per un vettore. */
bool simmetrico(vector<int> v)
{
 int i, j, len = v.size();
 for(i=0, j=len-1;i<j;i=i+1, j=j-1)
  {if (v[i]!=v[j]) {return false;}}
/*se il ciclo finisce, allora non ho trovato due valori
equidistanti dal centro del vettore e diversi*/
 return true;};
```

```
/* Es 19.3. inversione di un vettore */
void inverti(vector<int>& v)
{
 int i, j, len = v.size();
 for(i=0, j=len-1;i<j;i=i+1, j=j-1)
  {int temp = v[i]; v[i]=v[j]; v[j]=temp;}
/* per scambiare v[i], v[j] dobbiamo prima salvare v[i] in una
variabile temporanea temp.*/
};

/* Es. 19.4. esistenza di un elemento di un vettore v che soddisfa
una proprietà p */
bool esiste(vector<int> v, bool p(int x) )
{
 int i, len = v.size();
 for(i=0;i<len;i=i+1)
  {if (p(v[i])==true) {return true;}}
/* Se il ciclo finisce, allora non ho trovato un i tale che
p(v[i])=true */
 return false; /* la ricerca è fallita */
};

/* Es. 19.5. Cont di quanti elementi di un vettore v soddisfano
una proprietà p */
int conta(vector<int> v, bool p(int x) )
{
 int i, len = v.size(), count;
 for(i=0, count=0; i<len; i=i+1)
  {if (p(v[i])==true) {count=count+1;}}
/* Quando il ciclo finisce, count contiene il numero di i tali che
p(v[i])=true */
 return count; /* restituisco count */
};

/* Es 19.6. primo elemento che soddisfa una proprietà p */
int primoindice(vector<int> v, bool p(int x) )
{
 int i, len = v.size();
 for(i=0;i<len;i=i+1)
  {if (p(v[i])==true) {return i;}}
/* Se il ciclo finisce, allora non ho trovato un i tale che
p(v[i])=true */
 return -1; /* -1 non puo' essere un indice e rappresenta il
fallimento della ricerca */
};
```

```
bool p(int x) {return (x<0);}
bool q(int x) {return (x>2);}


int main(){
 vector<int> V(3,0), W(4,0), Z(3,0), T(7,0); /* definiamo v =
(1,2,1), w = (1,2,2,1), z = (1,2,2), T = (1,2,3,-3,2,1,10) */
 V[0]=1;V[1]=2;V[2]=1;
 W[0]=1;W[1]=2;W[2]=2;W[3]=1;
 Z[0]=1;Z[1]=2;Z[2]=2;
 T[0]=1;T[1]=2;T[2]=3;T[3]=-3;T[4]=2;T[5]=1;T[6]=10;

 cout << "Es. 19.1. Test di uguaglianza di due vettori" << endl;
 printVett(V,W,Z,T);
 cout << "uguali(V,V) = " << uguali(V,V) << endl;
 cout << "uguali(V,W) = " << uguali(V,W) << endl;
 cout << "uguali(W,Z) = " << uguali(W,Z) << endl;
 system("pause");system("CLS");

 cout << "Es 19.2. Test di simmetria di un vettore" << endl;
 printVett(V,W,Z,T);
 cout << "simmetrico(V)  = " << simmetrico(V) << endl;
 cout << "simmetrico(W)  = " << simmetrico(W) << endl;
 cout << "simmetrico(Z)  = " << simmetrico(Z) << endl;
 cout << "simmetrico(T)  = " << simmetrico(T) << endl;
 system("pause");system("CLS");

 cout << "Es. 19.3. Inversione di un vettore " << endl;
 printVett(V,W,Z,T);
 cout << "Invertiamo i quattro vettori" << endl;
 inverti(V);inverti(W);inverti(Z);inverti(T);
 printVett(V,W,Z,T);
 cout << "Invertiamo i quattro vettori una seconda volta" << endl;
 inverti(V);inverti(W);inverti(Z);inverti(T);
 printVett(V,W,Z,T);
 system("pause");system("CLS");

 cout << "Es. 19.4. Esistenza di un elemento che soddisfa la
proprietà" << endl << "p = essere negativi" << endl << "q = essere
>2" << endl;
 printVett(V,W,Z,T);
 cout << "esiste(V,p)  = " << esiste(V,p) << endl;
 cout << "esiste(W,p)  = " << esiste(W,p) << endl;
```

```
 cout << "esiste(Z,p)  = " << esiste(Z,p) << endl;
 cout << "esiste(T,p)  = " << esiste(T,p) << endl;
 cout << "esiste(V,q)  = " << esiste(V,q) << endl;
 cout << "esiste(W,q)  = " << esiste(W,q) << endl;
 cout << "esiste(Z,q)  = " << esiste(Z,q) << endl;
 cout << "esiste(T,q)  = " << esiste(T,q) << endl;
 system("pause");system("CLS");

 cout << "Es. 19.5. Conto del numero di elementi che soddisfano la
proprietà" << endl << "p = essere negativi" << endl << "q = essere
>2" << endl;
 printVett(V,W,Z,T);
 cout << "conta(V,p)  = " << conta(V,p) << endl;
 cout << "conta(W,p)  = " << conta(W,p) << endl;
 cout << "conta(Z,p)  = " << conta(Z,p) << endl;
 cout << "conta(T,p)  = " << conta(T,p) << endl;
 cout << "conta(V,q)  = " << conta(V,q) << endl;
 cout << "conta(W,q)  = " << conta(W,q) << endl;
 cout << "conta(Z,q)  = " << conta(Z,q) << endl;
 cout << "conta(T,q)  = " << conta(T,q) << endl;
 system("pause");system("CLS");

 cout << "Es. 19.6. Indice primo elemento che soddisfa la
proprietà" << endl << "p = essere negativi" << endl << "q = essere
>2" << endl;
 printVett(V,W,Z,T);
 cout << "primoindice(V,p)  = " << primoindice(V,p) << endl;
 cout << "primoindice(W,p)  = " << primoindice(W,p) << endl;
 cout << "primoindice(Z,p)  = " << primoindice(Z,p) << endl;
 cout << "primoindice(T,p)  = " << primoindice(T,p) << endl;
 cout << "primoindice(V,q)  = " << primoindice(V,q) << endl;
 cout << "primoindice(W,q)  = " << primoindice(W,q) << endl;
 cout << "primoindice(Z,q)  = " << primoindice(Z,q) << endl;
 cout << "primoindice(T,q)  = " << primoindice(T,q) << endl;
 system("pause");system("CLS");

}
```

Lezione 20: esercizi sui vettori
Parte 1. Esercizi

- Si consiglia di utilizzare un ciclo **for**.
- Per stampare un vettore di interi usate la funzione:

```
void printVett(vector<int> V){int i, len=V.size();
for(i=0;i<len;i=i+1){cout << V[i] << " ";}; cout << endl;}
```

- In alcuni esercizi di questa lezione utilizziamo come esempi i seguenti vettori di interi:
  V=(1,2,1), W=(1,2,2,1), Z=(1,2,2), T=(1,2,3,-3,2,1)

Es 20.1 (Inclusione insiemistica). Definite una funzione

bool incluso(**vector <int>** v, **vector<int>** w)

che prenda due vettori v, w, anche di lunghezze differenti, e restituisca **true** se ogni elemento di v compare in w, anche se in posizioni diverse e un numero diverso di volte, e **false** altrimenti. Per esempio (1,1,1) è incluso in (1). **Suggerimento.** Riutilizzate la funzione

bool appartiene(**int** x, **vector<int>** v)

vista in precedenza, che controlla l'appartenenza di un elemento a un vettore. Esplorate tutto il vettore v con un ciclo, e per ogni v[i] controllate se v[i] compare in w. Non appena trovate un v[i] che non compare in w, restituite **false** Se finite il ciclo, allora ogni elemento in v si trova anche in w. In questo caso, restituite **true. Esempi.** Se V, W, Z, T sono come sopra, allora incluso(W,V) = incluso(Z,V) = 1, mentre incluso(T,V) = 0, dato che 3 e -3 non compaiono in V.

Es 20.2 (Eguaglianza insiemistica). Definite una funzione

bool equiesteso(**vector <int>** v, **vector<int>** w)

che prenda due vettori v, w, anche di lunghezze differenti, e restituisca true se e solo se ogni elemento di v compare in w e viceversa, anche se i due elementi compaiono in posizioni diverse e un numero diverso di volte. **Suggerimento.** Riutilizzate la funzione dell'esercizio precedente che controlla l'inclusione. **Esempi.** Se V, W, Z, T sono come sopra, allora equiesteso(W,V) = equiesteso(Z,V) = 1, perché V, W, Z corrispondono all'insieme {1,2}, mentre equiesteso(T,V) = equiesteso(V,T) = 0, perché T corrisponde all'insieme {-3, 1, 2, 3}.

Es 20.3 (Il tipo delle informazioni geografiche) Definite un tipo

struct Informazioni {**vector<string>** naz, cap;}; **//ricordatevi il ;**

Definite I di tipo Informazioni, e inserite in I.naz una lista di nomi di nazioni, in I.cap una lista di nomi di capitali, della stessa lunghezza, tale che I.cap[i] sia la capitale della nazione I.naz[i], per ogni indice i. **Nota.** Quando definite I di tipo Informazioni, non fissate la lunghezza di I.naz e I.cap, quindi ***inizialmente entrambi sono vettori vuoti***. Dovete usare il comando

I.naz.push_back(s) per aggiungere s al vettore I.naz, e lo stesso per I.cap.

Es 20.3bis (Nazioni e Capitali). Scrivete una funzione

string capitale(**string** Naz, **Informazioni**& I)

La funzione prende un nome Naz di nazione, e restituisce il nome della corrispondente capitale, se la nazione compare in I.naz. Se la nazione non compare in I.naz, la funzione restituisce la stringa "non trovato". **Suggerimento.** Scandite il vettore I.cap alla ricerca di un i tale che I.naz[i]==Naz. Se lo trovate restituite I.cap[i]. Se terminate il ciclo senza averlo trovato, allora la nazione non compare sull'elenco. In questo caso restituite "non trovato". **Esempi.** Scrivete due righe di programma che chiedono una nazione e ne restituiscono la capitale, usando la funzione appena definita.

Es. 20.4 (vettore di frequenze). Scrivete una funzione

vector<int> istogrammaMCD(**int** n, **int** h)

che prende un intero n>0, e restituisce il vettore V delle frequenze del Massimo Comun Divisore per gli interi tra 1 e n. V ha h+1 elementi, e per ogni k=1,…,h: *V[k] è il numero di coppie (i,j) di interi tra 1 e n t tali che MCD(i,j)=k*.

Suggerimento. Riutilizzate la funzione MCD vista in precedenza. Definite un vettore V di h+1 elementi, inizialmente posti tutti uguali a 0. Utilizzate due cicli annidati, il piu' esterno per passare in rassegna tutte gli i=1,…,n, il piu' interno, per passare in rassegna tutti i j=1,…,n per ognuno di tali valori di i. Calcolate il valore k=MCD(i,j). Ogni volta che k<=h, ogni volta cioè che l'elemento V[k] fa parte di V, incrementate V[k] di 1.
Esempio. Sia n=10 mila, h=10, V=istogrammaMCD(n,h). Allora V[1], il numero di coppie di interi con MCD uguale a 1 *(cioè primi tra loro)* ed entrambi compresi tra 1 e 10 mila, vale: 60 794 971.

Es. 20.5 (Stima della probabilita' che MCD(i,j)=k). *(Questo esercizio non richiede di scrivere nuove funzioni ma solo di usare funzioni precedenti).* Stimate la probabilita' che MCD(i,j)=k, per k=1,2,3,…. **Suggerimento.** Riutilizzate la funzione istogrammaMCD dell'esercizio precedente per n = 10, 100, mille e 10 mila e per h=10. Sia Ist = istogrammaMCD(n,h). Definite un vettore **vector**<**double**> Freq di h elementi, tale che Freq[k] contiene la frequenza relativa con cui MCD(i,j)=k. Freq[k] vale per definizione Ist[k]/n^2 *(evitate di usare la divisione arrotondata tra interi: usate prima ((**double**) …) per convertire il divisore a numero reale).* Controllate le affermazioni fatte nella Presentazione del corso (§ 1), e cioè che: *(1)* Freq[1] vale circa $6/\pi^2$=0,6079 …, e *(2)* la frequenza con cui compare il valore k è proporzionale a $1/k^2$. Per verificare *(2)*, stampate i valori:

Freq[1]/Freq[1], Freq[2]/ Freq[1], Freq[3]/Freq[1], …

e controllate di ottenere all'incirca: 1, 1/4, 1/9, 1/16, … eccetera. Stampate gli inversi e otterrete: 1, 4, 9, 16, …

Lezione 20: esercizi sui vettori
Parte 2. Soluzioni

```
// Lezione20-Esercizi sui vettori.
#include <iostream>
#include <math.h>
#include <vector>
#include <stdlib.h>
using namespace std;

void printVett(vector<int> V)
 {int i, L=V.size();
  for(i=0;i<L;i=i+1){cout << V[i] << " ";}
  cout << endl;}

void printVett(vector<int> v, vector<int> w,
 vector<int> z, vector<int> t)
{cout << "vettore V: "; printVett(v);
 cout << "vettore W: "; printVett(w);
 cout << "vettore Z: "; printVett(z);
 cout << "vettore T: "; printVett(t);}

/* Es 20.1 (Inclusione insiemistica).
Definisco prima l'appartenenza. */
bool appartiene(int x, vector<int> v)
{int i, len=v.size();
 for(i=0; i<len; i=i+1)
  {
   if (v[i]==x)
    {return true;} //non appena trovo x restituisco i
  }
//se termino il ciclo senza trovare x, allora x non è in v
 return false;
};

bool incluso(vector <int> v, vector<int> w)
{int i, len=v.size();
 for(i=0;i<len;i=i+1)
  {if (appartiene(v[i],w)==false)
    {return false;}
/*non appena trovo un v[i] non w restituisco false*/
  }
/*se termino il ciclo allora ogni v[i] è in w*/
 return true;};
```

```
/* Es 20.2 (Eguaglianza insiemistica). */
bool equiesteso(vector <int> v, vector<int> w)
{return (incluso(v,w) && incluso(w,v));};

/* Es 20.3 (Nazioni e Capitali). */
 struct Informazioni {vector<string> naz, cap;}; //ricordatevi il ;

/* Informazioni è una struttura dati che contiene nomi di nazioni
e delle loro capitali. */
string capitale(string Naz, Informazioni& I)
{int i, len = I.naz.size();
 for(i=0;i<len;i=i+1)
  {if (Naz == I.naz[i])
    {return I.cap[i];}}
/* se alla fine del ciclo non ho trovato N nell'elenco
delle nazioni */
 return "non trovato";}

/* Es. 20.4 (vettore di frequenze). */
int MCD(int a, int b)//Siano a>0, b>=0
{while(a>0)
 {int v1 = b%a;
  int v2 = a;
      a  = v1;
      b  = v2;}
 return b;};

vector<int> istogrammaMCD(int n, int h)
{vector<int> I(h+1,0); int i, j;
 for(i=1;i<=n;i=i+1)
  {for(j=1;j<=n; j=j+1)
    {int m = MCD(i,j);
     if (m<=h) {I[m]=I[m]+1;}}}
 return I;
};

int main(){int i;
 vector <int> V(3,0), W(4,0), Z(3,0), T(6,0);
/* Definiamo v = (1,2,1), w = (1,2,2,1), z=(1,2,2), T=(1,2,3,-
3,2,1) */
 V[0]=1;V[1]=2;V[2]=1;
 W[0]=1;W[1]=2;W[2]=2;W[3]=1;
 Z[0]=1;Z[1]=2;Z[2]=2;
 T[0]=1;T[1]=2;T[2]=3;T[3]=-3;T[4]=2;T[5]=1;
```

```
 Informazioni I; /* Inizialmente i vettori I.naz e I.cap sono
vuoti. Aggiungo con il comando .push_back(.) nomi di nazioni e di
capitali alla struttura dati I */
 I.naz.push_back("Italia");       I.cap.push_back("Roma");
 I.naz.push_back("Francia");      I.cap.push_back("Parigi");
 I.naz.push_back("Inghilterra"); I.cap.push_back("Londra");
 I.naz.push_back("Germania");     I.cap.push_back("Berlino");
 I.naz.push_back("Spagna");       I.cap.push_back("Madrid");

 cout << "Es. 20.1. Test di inclusione insiemistica." << endl;
 printVett(V,W,Z,T);
 cout << "incluso(W,V) = " << incluso(W,V) << endl;
 cout << "incluso(Z,V) = " << incluso(Z,V) << endl;
 cout << "incluso(T,V) = " << incluso(T,V) << endl;
 system("pause");system("CLS");

 cout << "Es. 20.2. Test di uguaglianza insiemistica" << endl;
 printVett(V,W,Z,T);
 cout << "equiesteso(W,V) = " << equiesteso(W,V) << endl;
 cout << "equiesteso(Z,V) = " << equiesteso(Z,V) << endl;
 cout << "equiesteso(T,V) = " << equiesteso(T,V) << endl;
 system("pause");system("CLS");

 cout << "Es. 20.3. Nazioni e Capitali" << endl;
 cout << "Nazione  = ";
 string Naz; cin >> Naz;
 cout << "Capitale = " << capitale(Naz,I) << endl;
 system("pause");system("CLS");

 cout << "Es. 20.4. Istogramma del MCD" << endl;
 int n=10000, h=10;
 cout << " n = " << n << " h = " << h << endl;
 cout << " istogrammaMCD(n,h) = " << endl;
 vector<int> Ist = istogrammaMCD(n,h);
 for(i=1;i<=h;i=i+1)
  {cout << "coppie con MCD = "
    << i << "\t n. " << Ist[i] << endl;}
 system("pause");system("CLS");

 cout << "Es. 20.5. Stima della probabilita' del MCD" << endl;
 cout << "-----------------------------------------" << endl;
 vector<double> Freq(h+1);
 for(i=1;i<=h;i=i+1)
  {Freq[i] = Ist[i]/(double) (n*n);}
```

```
 for(i=1;i<=h;i=i+1)
  {cout << " Frequenza coppie con MCD = "
    << i << "\t" << Freq[i] << endl;}
 cout << "----------------------------------------" << endl;
 for(i=1;i<=h;i=i+1)
  {cout << " Frequenza relative coppie con MCD = "
    << i << "\t" << Freq[i]/Freq[1] << endl;}
 cout << "----------------------------------------" << endl;
 for(i=1;i<=h;i=i+1)
  {cout << " Inverso frequenza relative coppie con MCD = "
    << i << "\t" << round(Freq[1]/Freq[i]) << endl;}
 system("pause");system("CLS");
}
```

Tutorato 10 - Vettori
Parte 1. Esercizi proposti

Tut10.1. Scrivete una funzione

int Massimo(vector<int> v){}

che restituisce il massimo valore nel vettore v. **Esempio:** Massimo({1,2,1}) = 2.

Tut10.2. Scrivete una funzione

void SommeParziali(vector<int>& v){}

che modifica il vettore v inserendo al posto i la somma degli elementi di v di indice da 0 fino a i. **Esempio:** se v={v[0], v[1], v[2]} allora SommeParziali(v) = {v[0], v[0]+v[1], v[0]+v[1]+v[2]}.

Tut10.3. Scrivete una funzione

double Media(vector<int> v){}

che restituisce la media aritmetica degli elementi del vettore. **Esempio:** Media({1,2,3}) = (1+2+3)/3 = 2.

Tut10.4. Scrivete una funzione

vector<int> Prodotto(vector<int> v1, vector<int> v2){}

che restituisce un vettore w di lunghezza la minima tra le lunghezze di v1 e v2, in cui l'elemento di posto i è il prodotto dell'elemento di posto i di v1 e dell'elemento di posto i di v2. **Esempio:** Prodotto({1,2,3},{10,20,30}) = {10,40,90}.

Tut10.5. Scrivete una funzione

vector<int> Potenze(int n, int m){}

che restituisce il vettore di lunghezza m in cui l'elemento di posto i è n^i. **Esempio.** Potenze(10,4) = {1,10,100,1000}.

Provate le funzioni utilizzando il **main** seguente:

```
/* Tutorato 10 */
#include <iostream>
#include <math.h>
#include <stdlib.h>
#include <vector>
using namespace std;

/* Tut10.1 */
int Massimo(vector<int> v){};

/* Tut10.2 */
double Media(vector<int> v){};

/* Tut10.3 */
void SommeParziali(vector<int>& v){};
```

```
/* Tut10.4 */
vector<int> Prodotto(vector<int> v1, vector<int> v2){};

/* Tut10.5 */
vector<int> Potenze(int n, int m){};


   void printVett(vector<int> V)
   {int i, L=V.size();
    for(i=0;i<L;i=i+1){cout << V[i] << " ";}
    cout << endl;}

   void printVett(vector<int>W, vector<int> Z, vector<int> T)
   {cout << "W="; printVett(W);
    cout << "Z="; printVett(Z);
    cout << "T="; printVett(T);
    cout << endl;}

   int main(){vector <int> W(5,0), Z(5,0), T(5,0);
   W[0]=2; W[1]=4; W[2]=3; W[3]=1; W[4]=2;
   Z[0]=4; Z[1]=1; Z[2]=5; Z[3]=3; Z[4]=3;
   T[0]=1; T[1]=6; T[2]=4; T[3]=2; T[4]=3;

   printVett(W,Z,T);
   cout<< "Tut10.1 Massimo di Z=" << Massimo(Z) << endl;
   system("pause");system("cls");

   printVett(W,Z,T);
   cout << "Tut10.2. Somme Parziali di T = ";
   SommeParziali(T);printVett(T);
   system("pause");system("cls");

   printVett(W,Z,T);
   cout << "Tut10.3 Media di W=" << Media(W) <<endl;
   system("pause");system("cls");

   printVett(W,Z,T);
   cout << "Tut10.4 Prodotto di W e Z =";
   printVett(Prodotto(W,Z));
   system("pause");system("cls");

   cout << "Tut10.5 Potenze(10,4) = ";printVett(Potenze(10,4));
   system("pause");system("cls");}
```

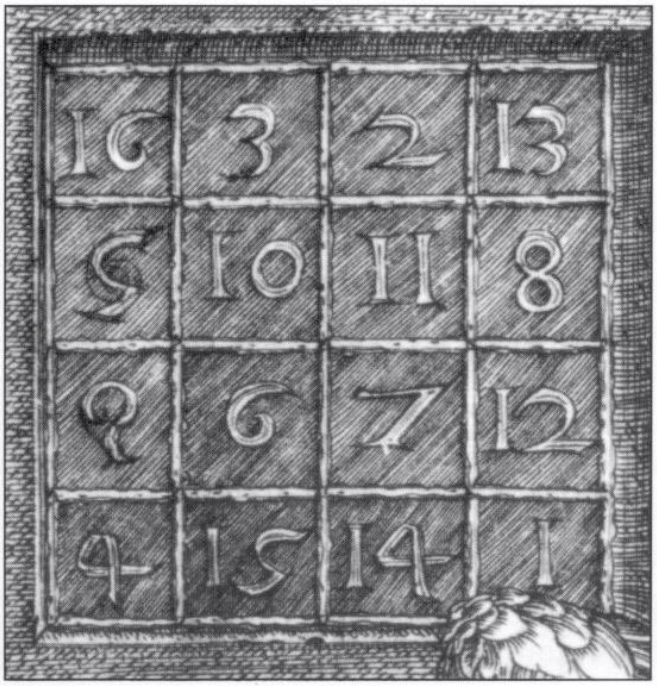
Tutorato 10: Vettori
Parte 2. Soluzioni

```
/* Tutorato 10 */
#include <iostream>
#include <math.h>
#include <stdlib.h>
#include <vector>
using namespace std;

/* Tut10.1 Massimo */
int Massimo(vector<int> v)
{int i, len=v.size(); int M=v[0];
 for(i=1; i<len; i=i+1)
  {if (M < v[i]) M = v[i];}
 return M;}

/* Tut10.2 Media */
double Media (vector <int> v)
{double somma=0; int i, len=v.size();
 for(i=0; i <len; i++)
  {somma=somma+v[i];}
 return somma/len;
}

/* Tut10.3 SommeParziali */
void SommeParziali (vector<int>& v)
{int i, len = v.size();
 for (i = 1; i<len; i=i+1)
  {v[i] = v[i-1]+v[i];}
}

/* Tut10.4 Prodotto */
vector<int> Prodotto (vector<int> v1, vector<int> v2)
{int len= min(v1.size(),v2.size()); vector<int> p(len);
for (int i=0; i<len; i++)
 {p[i]=v1[i]*v2[i];}
return p;
}

/* Tut10.5 */
vector <int> Potenze (int n, int m)
{vector<int> v(m);
```

```
 v[0]=1;
 for(int i=1; i<m; i=i+1)
  {v[i]=v[i-1]*n;}
 return v;
}

void printVett(vector<int> V)
{int i, L=V.size();
 for(i=0;i<L;i=i+1){cout << V[i] << " ";}
 cout << endl;}

void printVett(vector<int>W, vector<int> Z, vector<int> T)
{cout << "W="; printVett(W);
 cout << "Z="; printVett(Z);
 cout << "T="; printVett(T);
 cout << endl;}

int main(){vector <int> W(5,0), Z(5,0), T(5,0);
W[0]=2; W[1]=4; W[2]=3; W[3]=1; W[4]=2;
Z[0]=4; Z[1]=1; Z[2]=5; Z[3]=3; Z[4]=3;
T[0]=1; T[1]=6; T[2]=4; T[3]=2; T[4]=3;

printVett(W,Z,T);
cout<< "Tut10.1 Massimo di Z=" << Massimo(Z) << endl;
system("pause");system("cls");

printVett(W,Z,T);
cout << "Tut10.2. Somme Parziali di T = ";
SommeParziali(T);printVett(T);
system("pause");system("cls");

printVett(W,Z,T);
cout << "Tut10.3 Media di W=" << Media(W) <<endl;
system("pause");system("cls");

printVett(W,Z,T);
cout << "Tut10.4 Prodotto di W e Z =";
printVett(Prodotto(W,Z));
system("pause");system("cls");

cout << "Tut10.5 Potenze(10,4) = ";printVett(Potenze(10,4));
system("pause");system("cls");}
```

Settimana 11 - Matrici
"Rettangoli di dati"

1. **Lezione 21: Introduzione delle Matrici.**
   La libreria *"Matrix.cpp"*
2. **Lezione 22: Esercizi sulle Matrici:**
   risoluzione di sistemi con la formula di Laplace.
3. **Tutorato 11: Matrici**



16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

Albert Dürer. Melencolia, 1514 ***(dettaglio, da Wikipedia).*** In questo quadrato di numeri, detto "quadrato magico", la somma di orizzontali, verticali e diagonali è sempre 34.

Lezione 21: Introduzione alle Matrici
Parte 1. Tipo Matrix e libreria *"Matrix.cpp"*

Dato un tipo T, il tipo delle matrici su T consiste di tutti i rettangoli di nxm elementi di T, per qualche numero naturale n, m. La classe ***"Matrix"*** consiste della famiglia **Matrix**<**int**>, **Matrix**<**double**>, …, di tipi di matrici per T=int, double, … . Per esempio il tipo Matrix<int> contiene la seguente matrice 2x2 su interi:

1	**2**
3	**4**

Per ognuno di questi tipi **Matrix**<**int**>, **Matrix**<**double**>, …, vengono fornite alcune operazioni di base. La classe Matrix non fa parte delle librerie standard del C++, ma è stata definita appositamente per questo corso (ne ringraziamo il Prof. Ugo de' Liguoro). Capire come Matrix viene definita non fa parte del corso: il modo con cui **Matrix** rappresenta le matrici è di solito irrilevante per chi le usa. ***Solo per curiosità***, accenniamo che **Matrix** rappresenta una matrice come un ***"vettore di righe".*** Se M è come sopra, diventa un vettore **M = {V1, V2}**, i cui elementi sono gli array **V1={1,2}, V2={3,4}**, che a loro volta rappresentano le righe di M.

Per poter utilizzare **Matrix** all'interno dei programmi bisogna prima ***scaricare il file*** **Matrix** nello stesso faldone in cui si sta scrivendo il programma, poi aggiungere alla lista delle librerie incluse, il cosiddetto "preambolo" del programma, il comando:

```
#include "Matrix.cpp"
```

Includiamo una copia della libreria più avanti. L'uso della libreria **Matrix** è simile a quella della libreria **vector**, come mostrato nel file **UsoMatrix.cpp**. Qualche esempio: supponiamo di voler creare una matrice di reali. Il comando:

```
Matrix<double> M(2, 3);
```

crea una matrice **2 x 3** di elementi di tipo **double** (di numeri reali), non "inizializzati", ovvero con un valore iniziale non prevedibile. Se si vuole che la matrice abbia ogni elemento uguale a 0 occorre invece scrivere:

```
Matrix<double> M(2, 3, 0.0);
```

Data una matrice M, possiamo conoscerne le dimensioni usando le espressioni **M.Row()** per il numero di righe, e **M.Col()** per il numero di colonne. Notate che i comandi ".Row()", ".Col()" si scrivono con l'argomento M prima del comando.
Gli indici di M variano da 0 a M.Row()-1 per le righe e tra 0 e M.Col()-1 per le colonne.

L'elemento di indice i,j di M, detto ***"entry"*** della matrice, si può sia leggere che modificare scrivendo M[i][j] e M[i][j]= … . Per esempio: M[i][j]=7 assegna 7 all'elemento di indice i,j di M.

Una matrice può essere costruita per copia di una matrice data: **Matrix**<**double**> A = B; oppure utilizzando un **vettore del C** *(vedremo tra un attimo cosa sono)* che ne elenchi gli elementi tutti di seguito. Per esempio, la matrice

1	**2**	**3**
4	**5**	**6**

viene elencata con vettore del C che contiene {1, 2, 3, 4, 5, 6}. I vettori del C non hanno comandi come .size(), .push_back(), e si definiscono con la dichiarazione **int[100] V;** per un vettore di interi di 100 elementi, anziche' con **vector** <**int**> V(100);

A differenza dei vettori del C++, però, i vettori del C possono venire inizializzati con liste di elementi Un esempio:

```
int Binit[6] = {1, 2, 3, 4, 5, 6};
```

Si può anche scrivere:

```
int Binit[] = {1, 2, 3, 4, 5, 6};
```

lasciando che sia il compilatore a contare quanti elementi ci sono in Binit. Definito Binit, si può inizializzare una matrice B di interi partendo da Binit, come segue:

```
Matrix<int> B(2, 3, Binit);
```

B è una matrice 2 x 3 la cui prima riga è {1, 2, 3} e la seconda è {4, 5, 6}.

Una matrice può essere passata ad una funzione sia per valore che per riferimento (attraverso l'operatore &) ed essere restituita da una funzione, esattamente come già visto per la classe **vector.** Nel file ***"Matrix.cpp"*** è stata implementata una funzione di stampa delle matrici, aggiungendo questa capacità all'operatore << (si dice: per ***"overloading"*** o "sovraccarico" dell'operatore <<). Ciò significa che se A è una matrice, per vederne una stampa per righe basta usare <<:

```
cout << A;
```

Naturalmente occorre che i valori degli elementi di A siano a loro volta stampabili.

La libreria Matrix.cpp

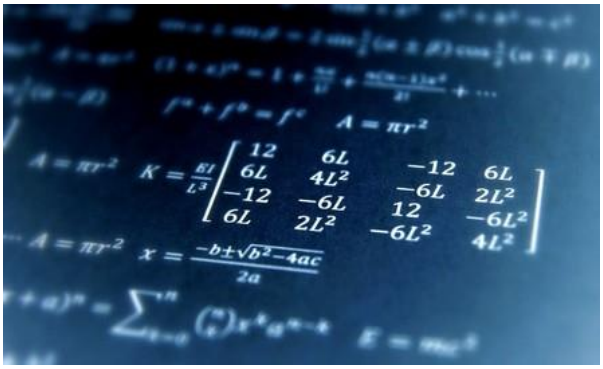
Qui di seguito includiamo una copia di ***"Matrix.cpp"***. Questa libreria viene usata nel corso ma non fa parte del corso.

```
/* Salvate le righe seguenti come il file "Matrix.cpp". Inserite
Matrix.cpp dentro lo stesso faldone in cui scrivete un programma
sulle matrici. Richiamatelo con #include "Matrix.cpp" */
```

```
#include <iostream>
#include <stdlib.h>
using namespace std;

template <class T>
class Matrix {
   private:
      int row, col;
      T** entr;
   public:
      inline Matrix();
      // costruttore di default
      Matrix(int r, int c);
      // costruttore di matrice rxc
      Matrix(int r, int c, T initval);
      // costruttore di matrice rxc con tutte le entrate = initval
      Matrix(int r, int c, T* array);
      // costruttore di matrice rxc con entrate da array[rxc]
elencate per righe
      Matrix(const Matrix<T> &m);
      // costruttore di copia da m
      ~Matrix();
      // deallocatore
      inline int Row() { return  row; }
      // ritona il numero delle righe
      inline int Col() { return col; }
      // ritona il numero delle colonne
      inline T*& operator[](int i) { return entr[i]; }
      //    consente    l'uso    della    notazione    m[i][j]    in
lettura/scrittura
      Matrix<T>& operator=(const Matrix<T> &m);
      // sovraccarico di =, permette di assegnare matrici a var.
di matrici
      template <class S>
      friend ostream& operator<<(ostream &os, const Matrix<S> &m);
      // stampa m per righe con il comando cout << m;
};

template <class T>
inline Matrix<T>::Matrix() {row = 0; col = 0; entr = NULL; }

template <class T>
Matrix<T>::~Matrix()
{
    for(int i = 0; i < row; i++)
```

```
        delete entr[i];
    delete entr;
}

template <class T>
Matrix<T>::Matrix(int r, int c)
{
      row = r; col = c;
      entr = new T*[row];
      for(int i = 0; i < row; i++)
          entr[i] = new T[col];
}

template <class T>
Matrix<T>::Matrix(int r, int c, T initval)
{
      row = r; col = c;
      entr = new T*[row];
      for(int i = 0; i < row; i++)
          entr[i] = new T[col];
      for(int i = 0; i < row; i++)
          for(int j = 0; j < col; j++)
              entr[i][j] = initval;
}


template <class T>
Matrix<T>::Matrix(int r, int c, T* array)
{
      row = r; col = c;
      entr = new T*[row];
      for(int i = 0; i < row; i++)
          entr[i] = new T[col];
      for(int i = 0; i < row; i++)
          for(int j = 0; j < col; j++)
              entr[i][j] = array[i*col + j];
}

template <class T>
Matrix<T>::Matrix(const Matrix<T> &m)
{
      row = m.row; col = m.col;
      entr = new T*[row];
      for(int i = 0; i < row; i++)
          entr[i] = new T[col];
```

```
        for(int i = 0; i < row; i++)
            for(int j = 0; j < col; j++)
                entr[i][j] = m.entr[i][j];
}

template <class T>
Matrix<T>& Matrix<T>::operator=(const Matrix<T> &m)
{
      for(int i = 0; i < row; i++)
         delete entr[i];
      delete entr;
      row = m.row; col = m.col;
      entr = new T*[row];
      for(int i = 0; i < row; i++)
           entr[i] = new T[col];
      for(int i = 0; i < row; i++)
           for(int j = 0; j < col; j++)
               entr[i][j] = m.entr[i][j];
      return *this;
}

template <class S>
ostream& operator<<(ostream &os, const Matrix<S> &m)
{
      for(int i = 0; i < m.row; i++)
      {
           for(int j = 0; j < m.col; j++)
               os << m.entr[i][j] << '\t';
           os << endl;
      }
      return os;
}

/* Fine libreria Matrix.cpp*/
```



Lezione 21: Introduzione alle Matrici
Parte 2. Esercizi

In questi esercizi dovete utilizzare la classe ***"Matrix"*** vista a lezione. Il file ***"Matrix.cpp"*** deve essere nella stessa cartella del file su cui state lavorando, e deve essere incluso nel vostro programma usando il comando:

#include "Matrix.cpp"

Dopo gli esercizi viene incluso un **main** di prova da utilizzare per ***controllare le vostre soluzioni su esempi***: ricopiatelo su un file .cpp. All'inizio il file contiene delle definizioni vuote per le funzioni richieste. Sta a voi rimpiazzare queste definizioni vuote con le vostre soluzioni, eseguire il main e controllare che i valori che ottenete siano quelli giusti.

Es. 21.1 (Massimo). Definite una funzione

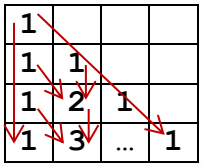
double Massimo(Matrix<double> M)

che data una matrice M reale di dimensione nxm, calcola il massimo degli elementi M. **Suggerimento.** Usate due cicli **for** annidati per esplorare tutti gli elementi M[i][j], per i=0,…,n-1 e j=0,…,m-1. Usate una variabile reale ris per contenere il risultato. ris parte da M[0][0], a ogni passo viene confrontata con M[i][j], e se risulta piu' piccolo viene rimpiazzata con M[i][j]. Finito il ciclo, restituite ris: ora contiene il massimo di M.

Es. 21.2 (Tartaglia). Definite una funzione

Matrix<int> Tartaglia(int n)

che restituisce come valore di ritorno la matrice di interi (n+1)x(n+1) il cui triangolo inferiore sinistro rappresenta le prime n+1 righe del triangolo di Tartaglia. **Suggerimento.** Partite da una matrice M di dimensione (n+1)x(n+1), assegnata tutta a 0. Assegnate prima M[i][0] e M[i][i] a 1, per i=0,…,n. Poi usate l'assegnazione M[i][j]=M[i-1][j-1]+ M[i-1][j] per ogni i=1,…,n e ogni j=1,…,i-1:



questa assegnazione traduce la definizione del Triangolo di Tartaglia. La matrice così ottenuta deve ancora essere stampata.

Es. 21.3 (Somma di Matrici). Definite una funzione

Matrix<double> somma(Matrix<double> &A, Matrix<double> &B)

che date due matrici A e B di uguali dimensioni nxm, restituisce come valore di ritorno la matrice somma C=A + B di dimensioni nxm. **Suggerimento.** Usate due cicli **for** annidati per esplorare tutti gli elementi A[i][j] e B[i][j], per i=0,…,n-1 e j=0,…,m-1. Usate una variabile C di matrice reale nxm per contenere il risultato. A ogni passo, C[i][j] viene assegnata a A[i][j] + B[i][j]. Finiti i due cicli annidati, restituite C: ora contiene A+B.

Es. 21.4 (Prodotto di Matrici). Definite una funzione
Matrix<double> prodotto(Matrix<double> &A, Matrix<double> &B)
che date le matrici reali A di dimensione nxm e B di dimensione mxp genera la matrice reale C di dimensione nxp, che rappresenta il prodotto riga per colonna C=A.B. **Suggerimento.** Usate una variabile C di matrice reale nxp per contenere il risultato. Usate due cicli **for** annidati per assegnare tutti gli elementi C[i][k], per i=0,…,n-1 e k=0,…,p-1. A ogni passo, C[i][k] viene assegnata alla somma di A[i][j]*B[j][k] per j=0, …, m-1, e cioè:
A[i][0]*B[0][k] + … + A[i][j]*B[j][k] + … + A[i][m-1]*B[m-1][k]
Per calcolare questa somma, usate un terzo ciclo **for** dentro i primi due, per j=0, …, m-1. Iniziate con C[i][k]=0.0 e a ogni passo aggiungete un addendo A[i][j]*B[j][k]. Finiti i tre cicli annidati, restituite C: ora contiene A.B.

```
/* Main di prova per gli esercizi della Lezione 21.
Ricopiate questo programma, completatelo e eseguitelo. Serve a
controllare su esempi le soluzioni che inserite. */
#include <iostream>
#include <stdlib.h>
#include "Matrix.cpp"
using namespace std;

Matrix<int> VuotaInt(0,0); /* matrice vuota di interi */
Matrix<double> VuotaDouble(0,0); /* matrice vuota di reali */

double Massimo(Matrix<double> M)
{return 0; /*sostituite con la soluzione*/ }

Matrix<int> Tartaglia(int n)
{return VuotaInt; /*sostituite con la soluzione*/ }

Matrix<double> somma(Matrix<double> &A, Matrix<double> &B)
{return VuotaDouble; /*sostituite con la soluzione*/ }

Matrix<double> prodotto(Matrix<double> &A, Matrix<double> &B)
{return VuotaDouble; /*sostituite con la soluzione*/ }

int main()
{   cout << "21.1 Massimo" << endl;
    double mm[] = { 3, -2, 2,
                    -1,  0, 4,
                    -4, -3, 1};
    Matrix<double> M(3, 3, mm);
```

```
    double max = Massimo(M);
    cout << max << " è il max di " << endl;
    cout << M << endl;
    system("pause");system("cls");

    cout << "21.2 Tartaglia" << endl; int n;
    cout << "Inserire l'ordine del Triangolo: "; cin >> n;
    Matrix<int> Tart = Tartaglia(n);
    cout << Tart << endl;
    system("pause");system("cls");

    cout << "21.3 Somma di Matrici" << endl;
    double a[] = {-1,  3, -2,
                   1,  0,  0,
                   1,  2,  2};
    Matrix<double> A(3, 3, a);
    cout << endl << "A=" << endl << A;
    double b[] = { 0,  0,  5,
                   2, -5,  0,
                   2,  1,  1};
    Matrix<double> B(3, 3, b);
    cout << endl << "B=" << endl << B;
    Matrix<double> C = somma(A, B);
    cout << endl << "somma(A, B)" << endl << C << endl;
    system("pause");system("cls");

    cout << "21.4 Prodotto di Matrici" << endl;
    double a2[] = { 1,  2,  3,
                    3,  4,  2};
    Matrix<double> A2(2, 3, a2);
    cout << endl << "A2=" << endl << A2;
    double b2[] = { 1,  4,  2,  6,
                    2,  1,  3,  4,
                    3, -1,  5,  2};
    Matrix<double> B2(3, 4, b2);
    cout << endl << "B2=" << endl << B2;
    Matrix<double> C2 = prodotto(A2, B2);
    cout << endl << "prodotto(A2, B2)=" << endl << C2 << endl;

/*   Dovreste ottenere la matrice:
14      3       23      20
17      14      28      38    */

    system("pause");system("cls");}
```

Lezione 21: Introduzione alle Matrici
Parte 3. Soluzioni

```
#include <iostream>
#include <stdlib.h>
#include "Matrix.cpp"
using namespace std;

double Massimo(Matrix<double> M)
{
    double max = M[0][0];
    for(int i = 0; i < M.Row(); i++)
     for(int j = 0; j < M.Col(); j++)
        if(M[i][j] > max)
            max = M[i][j];
    return max;
}

Matrix<int> Tartaglia(int n)
/* Supponiamo n >= 0. Allora la funzione restituisce una matrice
n+1 x n+1,  contenente le prime n+1 righe del triangolo di
Tartaglia */
{
    Matrix<int> Tr(n+1, n+1, 0);
    for(int i = 0; i <= n; i++)
    {   Tr[i][0] = 1;
        for(int j = 1; j <= i; j++)
            Tr[i][j] = Tr[i-1][j-1] + Tr[i-1][j];
    }
    return Tr;
}

Matrix<double> somma(Matrix<double> &A, Matrix<double> &B)
/* Supponiamo che A e B hanno le stesse dimensioni nxm. Allora la
funzione restituisce la matrice A + B */
{
     int n=A.Row(), m=A.Col(); Matrix<double> C(n, m);
     for(int i = 0; i < n; i++)
        for(int j = 0; j < m; j++)
            C[i][j] = A[i][j] + B[i][j];
     return C;
}

Matrix<double> prodotto(Matrix<double> &A, Matrix<double> &B)
/* Supponiamo A.Col() == B.Row(). Allora la funzione restituisce
il prodotto A.B "righe per colonne" */
```

```
{
    int n=A.Row(),m=A.Col(),p=B.Col(); Matrix<double> C(n,p,0.0);
    for(int i=0; i < n; i++)
     for(int k=0; k < p; k++)
      for(int j=0; j < m; j++)
       C[i][k]= C[i][k]+ A[i][j]*B[j][k];
    return C;
}

int main()
{   cout << "21.1 Massimo" << endl;
    double mm[] = { 3, -2, 2,
                    -1,  0, 4,
                    -4, -3, 1};
    Matrix<double> M(3, 3, mm);

    double max = Massimo(M);

    cout << max << " = max di " << endl;
    cout << M << endl;

    system("pause");system("cls");

    cout << "21.2 Tartaglia" << endl;
    int n;
    cout << "Inserire l'ordine del Triangolo: "; cin >> n;
    Matrix<int> Tart = Tartaglia(n);
    cout << Tart << endl;
    system("pause");system("cls");

    cout << "21.3 Somma di Matrici" << endl;
    double a[] = {-1,  3, -2,
                   1,  0,  0,
                   1,  2,  2};
    Matrix<double> A(3, 3, a);
    cout << endl << "A=" << endl << A;
    double b[] = { 0,  0,  5,
                   2, -5,  0,
                   2,  1,  1};
    Matrix<double> B(3, 3, b);
    cout << endl << "B=" << endl << B;
    Matrix<double> C = somma(A, B);
    cout << endl << "somma(A, B)" << endl << C << endl;
    system("pause");system("cls");
```

```
    cout << "21.4 Prodotto di Matrici" << endl;
    double a2[] = { 1,  2,  3,
                    3,  4,  2};
    Matrix<double> A2(2, 3, a2);
    cout << endl << "A2=" << endl << A2;
    double b2[] = { 1,  4,  2,  6,
                    2,  1,  3,  4,
                    3, -1,  5,  2};
    Matrix<double> B2(3, 4, b2);
    cout << endl << "B2=" << endl << B2;
    Matrix<double> C2 = prodotto(A2, B2);
    cout << endl << "prodotto(A2, B2)=" << endl << C2 << endl;
    /* stampa:
    14      3       23      20
    17      14      28      38
    */
    system("pause");system("cls");

}
```

Lezione 22: Matrici (risoluzione di sistemi) Parte 1. Formula di Laplace e Regola di Cramer

Questa è l'ultima esercitazione del corso: in seguito faremo solo simulazioni di esame. L'obbiettivo di questa esercitazione è ***piuttosto impegnativo:*** utilizzare la ***formula di Laplace*** per il calcolo del determinante di una matrice quadrata, e la ***regola di Cramer*** per la soluzione di un sistema lineare di n equazioni in n incognite. Cominciamo con un rapido cenno a questi due argomenti.

La Formula di Laplace

Sia data una matrice quadrata $A = (A_{ij})_{ij}$ di n righe e n colonne, dove A_{ij} è l'elemento nella riga i e colonna j. Sia $A^{(i,j)}$ la matrice che si ottiene da A eliminando la riga i e la colonna j: $A^{(i,j)}$ viene chiamato il minore complementare di (i,j). Se scegliamo un indice qualsiasi i di una riga, per esempio i=0, la formula di Laplace per il calcolo del determinante det(A) di A è:

$$\det(A) = \sum_{j=1}^{n} (-1)^{i+j} A_{i,j} \det(A^{(i,j)})$$

Si tratta di una definizione ***ricorsiva*** della funzione det(.), che il C++ è in grado di trasformare in un programma per il calcolo di det(.).

La Regola di Cramer

La regola di Cramer permette di calcolare, sebbene in modo ***molto inefficiente quando ci sono molte variabili***, la soluzione di un sistema di equazioni lineari ad *n* incognite ed *n* equazioni. Se *A* è la matrice quadrata *nxn* dei coefficienti e $c = (c_1, \ldots, c_n)$ il vettore colonna dei termini noti, il sistema è detto ***determinato*** se $\det(A) \neq 0$. In questo caso il vettore soluzione $x = (x_1, \ldots, x_n)$ è dato da:

$$x_j = \frac{\det A_j}{\det A}$$

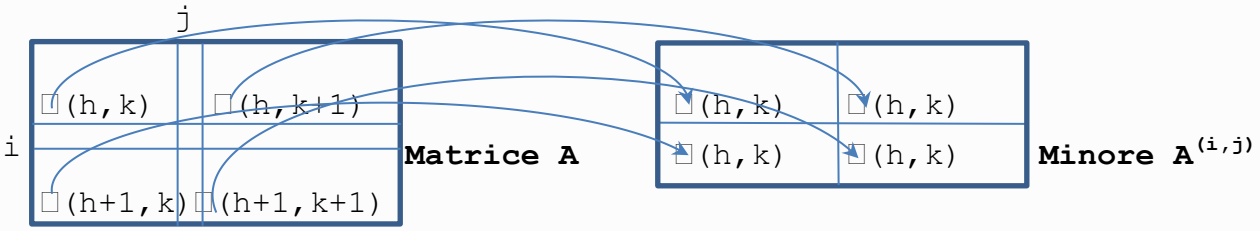
dove A_j è la matrice ottenuta da *A* sostituendo la colonna *j* con il vettore colonna *c*.

Per maggiori dettagli sulle formule di Cramer e Laplace si veda ad esempio Wikipedia.

Lezione 22: Matrici (risoluzione di sistemi)
Parte 2. Esercizi

In questa esercitazione vi forniamo un programma, contenente: le funzioni degli es. **22.1-22.4**, da completare, e un **main** con dei calcoli di prova, per aiutarvi a controllare le vostre soluzioni.

Es. 22.1 e 22.2. Si completi il file fornito qui sotto (che usa il file ***"Matrix.cpp"***) in modo da ottenere: ***(22.1)*** una funzione che calcola il minore complementare $A^{(i,j)}$, ricopiando i dati da A come indicato qui sotto:



(22.2) una funzione ***ricorsiva*** che calcola il determinante direttamente se n=1,2, altrimenti usando la regola di Laplace che abbiamo appena visto. **Suggerimento.** La funzione che calcola il minore complementare deve definire una nuova matrice, non modificare la matrice data come argomento, altrimenti i dati originari vengono subito persi. **Esempi.** Controllate i seguenti determinanti:

$$\begin{vmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \end{vmatrix} = 0$$

$$\begin{vmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 \\ 0 & 2 & 2 & 0 \end{vmatrix} = 2$$

$$\begin{vmatrix} 1 & 2 & 3 & 3 & 5 \\ 3 & 2 & 1 & 2 & 2 \\ 1 & 2 & 3 & 4 & 5 \\ -1 & 0 & -8 & 1 & 2 \\ 7 & 2 & 1 & 3 & 2 \end{vmatrix} = -224$$

Es 22.3 e 22.4. Si completi il programma fornito qui sotto in modo da ottenere: ***(22.3)*** una funzione che calcola la matrice A_j ottenuta da A sostituendo la colonna j con un vettore colonna c, e ***(22.4)*** una funzione che risolve un sistema lineare di n equazioni in n incognite, usando la regola di Cramer appena vista. **Suggerimento.** La funzione che calcola la matrice A_j deve definire una nuova matrice, non modificare la matrice A data come

argomento, altrimenti i dati originari vengono subito persi.
Esempi. Si controlli il programma scritto risolvendo il sistema:

$$\begin{cases} x_1 - x_2 + x_3 = 6 \\ 2x_1 + x_2 - x_3 = -3 \\ x_1 - x_2 - x_3 = 0 \end{cases}$$

di determinante = -6 e di soluzioni $x_1 = 1$, $x_2 = -2$ e $x_3 = 3$.
Nella pagine successive forniamo un programma da completare.

```
/* Lezione 22 - Programma da completare:
   Determinanti e regola di Cramer */
#include <iostream>
#include <stdlib.h>
#include <vector>
#include <math.h>
#include "Matrix.cpp"
using namespace std;

// Vettori e matrici banali
vector<double> VVuoto(0); Matrix<double> MVuota(0,0);

// Es. 22.1 Calcolo del minore complementare
Matrix<double> minor(Matrix<double> &M, int i, int j)
/* Supponiamo che  M è una matrice nxn e che 0<=i,j<n. Allora la
funzione restituisce il minore M^(i,j), ossia: M meno la riga i e
la colonna j */
{
     return MVuota; // soluzione NON corretta
}

// Es. 22.2 Calcolo del determinante con regola di Laplace
double det(Matrix<double> &M)
/* Supponiamo che  M è una matrice quadrata. Allora la funzione
restituisce il determinante di M */
{
     return 0; // soluzione NON corretta
}

// Es. 22.3 Sostituzione di colonna con vettore in una matrice
Matrix<double> Subst(Matrix<double> &A, vector<double> &C, int j)
/* Supponiamo: A matrice nxm, C un vettore di dimensione n, 0<=j<
m. Allora la funzione restituisce una copia di A in cui la colonna
j è sostituita dal vettore C */
{
     return MVuota; // soluzione NON corretta
}
```

```
// 22.4 Risoluzione di un sistemi di equazioni con Cramer
vector<double> Cramer(Matrix<double> &A, vector<double> &C)
/* Supponiamo: A matrice quadrata nxn, C vettore di dim. n. Allora
se detA non è 0, la funzione restituisce il vettore X tale che
AX=C, altrimenti restituisce il vettore "vuoto" (di 0 elementi) */
{
  return VVuoto; // soluzione NON corretta
}


int main() // main di prova, per sperimentare le vostre funzioni
{double a[] =
       { 1,  1,  0,  0,
         0,  1,  1,  0,
         1,  0, -1,  0,
         0,  1,  0,  1  };

     Matrix<double> A(4, 4, a);
     cout << "Matrice A" << endl;
     cout << A << endl;
     Matrix<double> AM = minor(A, 2, 1);
     cout << "Minore (2,1) di A = " << endl << AM << endl;
     cout << "det(A) = " << det(A) << endl << endl;

     double b[] =
       { 1,  0,  0,  1,
         0,  1,  0,  0,
         2,  1,  0,  1,
         0,  2,  2,  0  };

     Matrix<double> B(4, 4, b);

     cout << "Matrice B" << endl;
     cout << B << endl;
     cout << "det(B) = " << det(B) << endl << endl;

     double c[] =
     {   1,  2,  3,  3,  5,
         3,  2,  1,  2,  2,
         1,  2,  3,  4,  5,
        -1,  0, -8,  1,  2,
         7,  2,  1,  3,  2  };

     Matrix<double> C(5, 5, c);
```

```
     cout << "Matrice C" << endl;
     cout << C << endl;
     cout << "det(C) = " << det(C) << endl << endl;

/* Dato il sistema:
          x - y + z =  6
         2x + y - z = -3
          x - y - z =  0
ne calcoliamo le soluzioni col metodo di Cramer. */

// Sia S la matrice dei coefficienti del sistema:
     double s[] =
     {    1,  -1,  1,
          2,   1, -1,
          1,  -1, -1  };

     Matrix<double> S(3, 3, s);
     cout << "Matrice dei coefficienti:" << endl;
     cout << S << endl;

// Sia V il vettore colonna dei termini noti del sistema:
     vector<double> V(3);
     V[0] = 6;
     V[1] = -3;
     V[2] = 0;
     cout << "vettore dei termini noti" << endl;
     for(int i = 0; i < 3; i++) cout << V[i] << endl;

/* Allora X è il vettore delle soluzioni del sistema, se esistono
e sono uniche */
vector<double> X = Cramer(S, V);
if(X.size() == 0)
 cout << "Sistema non determinato" << endl;
for(int i = 0; i < X.size(); i++)
 cout << "Soluzione x_" << i+1 << " = " << X[i] << endl;

/* Il risultato dovrebbe essere:
      Soluzione x_1 = 1
      Soluzione x_2 = -2
      Soluzione x_3 = 3           */

    system("pause");}
```

Lezione 22: Matrici (risoluzione di sistemi)
Parte 3. Soluzioni

```
/* Lezione 22 - Soluzioni
   Determinanti, formula di Laplace e regola di Cramer */

#include <iostream>
#include <stdlib.h>
#include <vector>
#include "Matrix.cpp"
using namespace std;

// Es. 22.1 Calcolo del minore complementare
Matrix<double> minor(Matrix<double> &M, int i, int j)
/* Supponiamo che  M è una matrice nxn e che 0<=i,j<n. Allora la
funzione restituisce il minore M^(i,j), ossia: M meno la riga i e
la colonna j */
{    Matrix<double> A (M.Row()-1, M.Col()-1);
     for(int h = 0; h < A.Row(); h++)
        for(int k = 0; k < A.Col(); k++)
            if(h < i && k < j)
                A[h][k] = M[h][k];
            else if(h < i && k >= j)
                A[h][k] = M[h][k+1];
            else if(h >= i && k < j)
                A[h][k] = M[h+1][k];
            else // h >= i && k >= j
                A[h][k] = M[h+1][k+1];
     return A;
}

// Es. 22.2 Calcolo del determinante con regola di Laplace
double det(Matrix<double> &M)
/* Supponiamo che M è una matrice quadrata. Allora la funzione
restituisce il determinante di M */
{    if(M.Row() == 2) // caso di base
        return M[0][0]*M[1][1] - M[0][1]*M[1][0];
     else // caso generale
     {
        double s = 0.0;
        int segno = 1;
/* In base alla formula di Laplace per i=0, sommo le espressioni:
M[0][j]*det(minore complementare per (0,j)), per ogni j */
        for(int j = 0; j < M.Col(); j++)
        { Matrix<double> A = minor(M, 0, j);
          s = s + segno * M[0][j] * det(A);
```

```
            segno = - segno; //il segno e' alternativamente +1 e -1
        }
        return s;
    }
}


// Es. 22.3 Sostituzione di colonna con vettore in una matrice
Matrix<double> Subst(Matrix<double> &A, vector<double> &C, int j)
/* Supponiamo: A matrice nxm, C un vettore di dimensione n, 0<=j<
m. Allora la funzione restituisce una copia di A in cui la colonna
j è sostituita dal vettore C */
{
     Matrix<double> A_j = A;
     for(int i = 0; i < A.Row(); i++)
        A_j[i][j] = C[i];
     return A_j;
}

// 22.4 Risoluzione di un sistemi di equazioni con Cramer
vector<double> Cramer(Matrix<double> &A, vector<double> &C)
/* Supponiamo: A matrice quadrata nxn, C vettore di dim. n. Allora
se detA non è 0, la funzione restituisce il vettore X tale che
AX=C, altrimenti restituisce il vettore "vuoto" (di 0 elementi) */
{
    double det_A = det(A);
    int n = A.Row(); int m = A.Col(); int p = C.size();
    if ((n!=m)||(n!=p)||(det_A == 0))
// A è non-quadrata oppure singolare oppure C non ha n elementi
    {   vector<double> X(0); // Costruisco un vettore vuoto
        return X; // Restituisco un vettore vuoto
    }
    else
// A è non-singolare di nxn elementi e C ha n elementi
    {
     vector<double> X(n);// Costruisco un vettore di n elementi
     for(int j = 0; j < X.size(); j++)
            {
             Matrix<double> A_j = Subst(A, C, j);
             X[j] = det(A_j)/det_A;
            }
     return X;
    }
}
```

```
int main()
{    double a[] =
       { 1,  1,  0,  0,
         0,  1,  1,  0,
         1,  0, -1,  0,
         0,  1,  0,  1  };

     Matrix<double> A(4, 4, a);
     cout << "Matrice A" << endl;
     cout << A << endl;

     Matrix<double> AM = minor(A, 2, 1);
     cout << "Minore (2,1) di A = " << endl << AM << endl;

     cout << "det(A) = " << det(A) << endl << endl;

     double b[] =
       { 1,  0,  0,  1,
         0,  1,  0,  0,
         2,  1,  0,  1,
         0,  2,  2,  0  };

     Matrix<double> B(4, 4, b);
     cout << "Matrice B" << endl;
     cout << B << endl;
     cout << "det(B) = " << det(B) << endl << endl;

     double c[] =
     {   1,  2,  3,  3,  5,
         3,  2,  1,  2,  2,
         1,  2,  3,  4,  5,
        -1,  0, -8,  1,  2,
         7,  2,  1,  3,  2  };

     Matrix<double> C(5, 5, c);
     cout << "Matrice C" << endl;
     cout << C << endl;
     cout << "det(C) = " << det(C) << endl << endl;

/* Dato il sistema:
          x - y + z =  6
         2x + y - z = -3
          x - y - z =  0
ne calcoliamo le soluzioni col metodo di Cramer. */
```

```
// S matrice dei coefficienti
     double s[] =
     {    1,  -1,  1,
          2,   1, -1,
          1,  -1, -1  };

     Matrix<double> S(3, 3, s);
     cout << "Matrice dei coefficienti:" << endl;
     cout << S << endl;

// V è il vettore colonna dei termini noti
     vector<double> V(3);
     V[0] = 6;
     V[1] = -3;
     V[2] = 0;
     cout << "vettore dei termini noti" << endl;
     for(int i = 0; i < 3; i++)
          cout << V[i] << endl;

// X è il vettore delle soluzioni, se esistono
     vector<double> X = Cramer(S, V);
     if(X.size() == 0)
          cout << "Sistema non determinato" << endl;
     for(int i = 0; i < X.size(); i++)
          cout << "Soluzione x_" << i+1 << " = " << X[i] << endl;

/*
      Il risultato dovrebbe essere:
      Soluzione x_1 = 1
      Soluzione x_2 = -2
      Soluzione x_3 = 3
*/
     system("pause");}
```

Tutorato 11: Matrici
Parte 1. Esercizi proposti

1	4	8	8	3	3	1	6	8	4
9	1	1	3	4	4	9	6	1	6
9	9	4	4	2	3	7	6	1	5
7	0	1	8	7	2	0	8	8	5
3	9	1	5	8	1	8	0	2	5
9	3	9	7	1	5	2	8	2	6
5	5	4	7	4	2	5	3	1	4
9	7	1	0	3	7	2	7	8	6
7	1	1	8	7	4	5	9	8	2
1	2	3	7	4	9	0	9	0	3

Una matrice casuale 10 x 10 di cifre da 0 a 9: esistono 1 googol = 10^{100} matrici di questo tipo

In questi esercizi dovete utilizzare la classe ***"Matrix.cpp"*** vista a lezioni. Il file ***"Matrix.cpp"*** deve essere nella stessa cartella del file su cui state lavorando, e deve essere incluso nel vostro programma usando il comando:

#include "Matrix.cpp"

Dopo gli esercizi viene incluso un **main di prova** da utilizzare per controllare le soluzioni. Il vostro compito è completare le funzioni lasciate incomplete e elencate qui sotto.

Tut11.1 Definite una funzione

Matrix<int> TavolaPitagorica(int n)

che dato n costruisce la matrice di n righe e n colonne, che alla riga i e colonna j contiene il prodotto (i+1)*(j+1). **Esempio.** Se stampate TavolaPitagorica(10) ottenete la tavola pitagorica.

Tut11.2 Definite una funzione

Matrix<int> PariDispari(int n, int m)

che dato n costruisce la matrice di n righe e m colonne, che alla riga i e colonna j contiene 0 se i+j è pari, e contiene 1 se i+j è dispari. **Esempio.** PariDispari(2) = {{0,1},{1,0}}.

Tut11.3 Definite una funzione

Matrix<int> Trasposta(Matrix<int> M)

che prende una matrice M di dimensioni nxm qualsiasi, e restituisce la matrice N trasposta della matrice data. **Nota.** N ha dimensioni mxn e per ogni posizione i,j contiene l'elemento di posto j,i di M. **Esempio.** Se M = {{1,2}, {3,4}}, allora

Trasposta(M) = {{1,3}, {2,4}}.

Tut11.4 Definite una funzione

bool Uguali(Matrix<int> M, Matrix<int> N)

che prende due matrici M, N, e restituisce true se le due matrici sono uguali (hanno le stesse dimensioni e gli stessi elementi in ogni posizione), e false altrimenti. **Suggerimento.** Controllate che M, N abbiano lo stesso numero di righe e lo stesso numero di colonne. Se è così usate un ciclo per confrontare ogni elemento di M con l'elemento corrispondente di N. Non appena trovate due elementi diversi restituite false. Se arrivate alla fine del ciclo senza aver trovato elementi diversi, allora M=N: restituite true. **Esempio.** Se M = {{1,2}, {3,4}} e N = {{1,3}, {2,4}} allora Uguali(M,M)=true e Uguali(M,N) = false.

Tut11.5 Definite una funzione

bool Simmetrica(Matrix<int> M)

che prende una matrice M, e restituisce true se la matrice è simmetrica (se M è quadrata e l'elemento di posto i,j è sempre uguale all'elemento di posto j,i), e false altrimenti. **Suggerimento.** Usate la soluzione degli esercizi precedenti e confrontate M con Trasposta(M). **Esempio.** Se M = {{1,2}, {3,4}} allora Simmetrica(M) = false e se N = TavolaPitagorica(10) allora Simmetrica(N) = true.

```
/*                      MAIN DI PROVA
 Dovete utilizzare il main di prova per controllare le vostre
soluzioni su esempi. Le funzioni incluse qui sotto sono tutte da
completare. */

#include <iostream>
#include "Matrix.cpp"
using namespace std;
Matrix<int> Vuota(0,0);

/* Tut11.1. Tavola Pitagorica (da completare) */
Matrix<int> TavolaPitagorica(int n)
{return Vuota;}

/* Tut11.2. Pari e Dispari (da completare)  */
Matrix<int> PariDispari(int n, int m)
{ return Vuota;}

/* Tut11.3 Trasposta (da completare) */
Matrix<int> Trasposta(Matrix<int> M)
{return Vuota;}
```

```
/* Tut11.4 Uguali (da completare) */
bool Uguali(Matrix<int> M, Matrix<int> N)
{}

/* Tut11.5 Simmetrica (da completare) */
bool Simmetrica(Matrix<int> M)
{}



int main(){Matrix<int> M(2,2);
 M[0][0]=1;M[0][1]=2;M[1][0]=3;M[1][1]=4;

 cout << "Tut11.1 TavolaPitagorica \n TavolaPitagorica(5)="
<< endl << TavolaPitagorica(5) << endl;
 system("PAUSE");system("cls");

 cout << "Tut11.2 Pari e Dispari \n PariDispari(2,3)=" << endl
  << PariDispari(2,3) << endl;
 system("PAUSE");system("cls");

 cout << "Tut11.3 Trasposta \n M=" << endl << M << endl;
 cout << "Trasposta(M)=" << endl <<
  Trasposta(M) << endl;
 system("PAUSE");system("cls");

 cout << "Tut11.4 Uguali \n M=" << endl << M << endl;
 cout << "Uguali(M,M)=" << endl <<
  Uguali(M,M) << endl << endl;
 cout << "Uguali(M,Trasposta(M))=" << endl <<
  Uguali(M,Trasposta(M)) << endl << endl;
 cout << "Uguali(PariDispari(2,2),PariDispari(2,3))=" << endl
  << Uguali(PariDispari(2,2),PariDispari(2,3)) << endl << endl;
 system("PAUSE");system("cls");

 cout << "Tut11.5 Simmetrica \n M=" << endl << M << endl;
 cout << "Simmetrica(M)=" << endl <<
  Simmetrica(M) << endl << endl;
 cout << "PariDispari(2,2)=" << endl <<
  PariDispari(2,2) << endl << endl;
 cout << "Simmetrica(PariDispari(2,2))=" << endl <<
  Simmetrica(PariDispari(2,2)) << endl << endl;
 system("PAUSE");system("cls");}
```

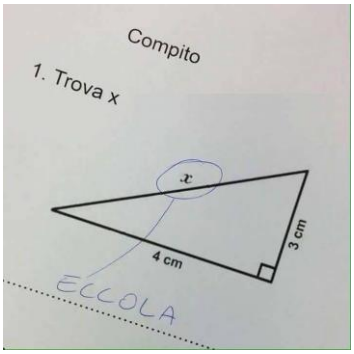
Tutorato 11 - Matrici
Parte 2. Soluzioni

```
#include <iostream>
#include "Matrix.cpp"
using namespace std;

/* Tut11.1. Tavola Pitagorica */
Matrix<int> TavolaPitagorica(int n)
{int i, j; Matrix <int> M(n,n);
 for (i=0; i<n; i++)
  {for (j=0; j<n; j++)
    {M[i][j]=(i+1)*(j+1);}
  }
 return M;
}

/* Tut11.2. Pari e Dispari. */
Matrix<int> PariDispari(int n, int m)
{int i, j; Matrix <int> M(n,m);
 for (i=0; i<n; i++)
  {for (j=0; j<m; j++)
    {M[i][j]=(i+j)%2;}
  }
 return M;
}

/* Tut11.3 Trasposta */
Matrix<int> Trasposta(Matrix<int> M)
{int i, j, n=M.Row(), m=M.Col(); Matrix <int> N(m,n);
 for (i=0; i<n; i++)
  {for (j=0; j<m; j++)
    {N[j][i]=M[i][j];}
  }
 return N;
}

/* Tut11.4 Uguali */
bool Uguali(Matrix<int> M, Matrix<int> N)
{int i, j, n=M.Row(), m=M.Col();
 if ((n!=N.Row()) || (m!=N.Col()))
  return false;
 for (i=0; i<n; i++)
  {for (j=0; j<m; j++)
```

```
    {if (M[i][j]!=N[i][j])
      return false;}
  }
 return true;
}

/* Tut11.5 Simmetrica*/
bool Simmetrica(Matrix<int> M)
{return Uguali(M,Trasposta(M));}


int main(){Matrix<int> M(2,2);
 M[0][0]=1;M[0][1]=2;M[1][0]=3;M[1][1]=4;

 cout << "Tut11.1 TavolaPitagorica \n TavolaPitagorica(5)="
<< endl << TavolaPitagorica(5) << endl;
 system("PAUSE");system("cls");

 cout << "Tut11.2 Pari e Dispari \n PariDispari(2,3)=" << endl
  << PariDispari(2,3) << endl;
 system("PAUSE");system("cls");

 cout << "Tut11.3 Trasposta \n M=" << endl << M << endl;
 cout << "Trasposta(M)=" << endl <<
  Trasposta(M) << endl;
 system("PAUSE");system("cls");

 cout << "Tut11.4 Uguali \n M=" << endl << M << endl;
 cout << "Uguali(M,M)=" << endl <<
  Uguali(M,M) << endl << endl;
 cout << "Uguali(M,Trasposta(M))=" << endl <<
  Uguali(M,Trasposta(M)) << endl << endl;
 cout << "Uguali(PariDispari(2,2),PariDispari(2,3))=" << endl
  << Uguali(PariDispari(2,2),PariDispari(2,3)) << endl << endl;
 system("PAUSE");system("cls");

 cout << "Tut11.5 Simmetrica \n M=" << endl << M << endl;
 cout << "Simmetrica(M)=" << endl <<
  Simmetrica(M) << endl << endl;
 cout << "PariDispari(2,2)=" << endl <<
  PariDispari(2,2) << endl << endl;
 cout << "Simmetrica(PariDispari(2,2))=" << endl <<
  Simmetrica(PariDispari(2,2)) << endl << endl;
 system("PAUSE");system("cls");}
```

Settimana 12
Preparazione all'Esame

1. **Lezione 23: Preparazione all'esame**
2. **Lezione 24: Preparazione all'esame**
3. **Tutorato 12: Matrici (soluzione di sistemi)**



Lezione 23: Preparazione all'esame
Parte 1. Esercizi

In questa lezione assegniamo quattro esercizi di difficolta' simile a quelli che saranno dati all'esame del corso. Per ogni esercizio viene dato un **main** di prova che aiuta a controllare se la vostra soluzione è corretta. Per quattro esercizi all'esame vengono date due ore: qui diamo invece 90 minuti, e useremo gli ultimi 30 minuti della lezione per la correzione. Facciamo assistenza alla prova ma non aiutiamo a risolvere gli esercizi.

Gli esercizi di questa lezione riguardano stringhe, calcolo numerico, vettori, e ricorsione. Nella prossima lezione vedremo quattro esercizi su matrici, strutture e ricorsione.

Es. 23.1. Trasformazione in minuscolo. Scrivete una funzione

string stringToLower(**string** s)

che data una stringa s ne trasformi in minuscolo tutti i caratteri alfabetici maiuscoli. **Suggerimento.** Il carattere minuscolo corrispondente alla maiuscola c si calcola con la funzione **char** tolower(**char** c), che fa parte del C++. Se c non è una maiuscola allora tolower(c) = c. **Esempio.** Sia s = "ABCxyz123DEF". Allora stringToLower(s) = "abcxyz123def".

```
/* Es. 23.1 Trasformazione in minuscolo. Testo */
#include <iostream>
#include <math.h>
#include <stdlib.h>
using namespace std;

string stringToLower(string s)
 { return ""; /* Rimpiazzate return ""; con la soluzione */ }

int main() {string s;
 cout << "Inserite una stringa con maiuscole:" <<endl;
 cin >> s;
 cout << "La stessa stringa in minuscole:" << endl <<
     stringToLower(s)  << endl;
 system("PAUSE"); }
```

Es. 23.2 (La sommatoria L). Scrivete una funzione

double L (double x, int n)

per calcolare la somma dei termini di grado da 1 a n della serie il cui termine di grado i vale: $(-(-x)^i)/i$. **Nota.** Se i=1 allora $(-(-x)^i)/i$ vale x, se i=2 allora $(-(-x)^i)/i$ vale $-x^2/2$. Dunque la funzione L(x,n) vale:

$L(x,n) = x - x^2/2 + x^3/3 - x^4/4 + ... + \boldsymbol{(-(-x)^n)/n}$

Per calcolare la potenza usate la funzione pow(a,b) della libreria "math" *(non usate a^b, non è la potenza)*. **Esempi.** L(0.5, 3) = 0.41667…, L(0.5, 10) = 0.405435 ….

```
/* Es. 23.2 La sommatoria L. Testo */
#include <iostream>
#include <math.h>
#include <stdlib.h>
using namespace std;

double L (double x, int n){ /* Inserite qui la soluzione */ }

int main(){cout << "L(0.5,3)=" << L(0.5,3) << endl << " dovrebbe valere circa 0.41667" << endl;
 cout << "L(0.5,10)=" << L(0.5,10) << endl << " dovrebbe valere circa 0.405435" << endl;
 system("PAUSE");}
```

Es 23.3 (Meno Frequente). Definite una funzione

int MenoFrequente(vector<int> v)

che prenda in input un vettore non vuoto v di interi. La funzione restituisce uno dei valori che compaiono il minor numero di volte in v, tra quelli che compaiono almeno una volta. **Esempio.** Se v = ***{1, 2, 3, 2, 1, 1, 5, 2, 3, 2, 5}*** allora MenoFrequente(v) = 3 oppure 5, dato che questi interi compaiono due volte, mentro gli altri interi 1, 2 compaiono tre o quattro volte. **Suggerimento.** Definite una funzione **int** Conta(**vector<int>** v, **int** x) che conta quante volte x compare in v. Nella definizione di MenoFrequente, usate una variabile reale m per contenere il risultato. m parte da v[0]. Usate "Conta" per contare quante volte m=v[0] compare in v, poi quante volte v[1] compare in v. Se v[1] compare meno volte di m=v[0], assegnate a m il valore v[1]. Usate "Conta" per contare quante volte v[2] compare in v, e confrontatelo con il numero di volte che m compare in v. Se v[2] compare meno volte di m, assegnate a m il valore v[2]. Continuate così finche' non avete percorso tutto v. Il valore finale di m è uno dei valori che compare il più basso numero di volte in v, tra quelli che compaiono almeno una volta. Per la precisione, è il più a sinistra tra di essi. Nell'esempio sopra, il più a sinistra tra i meno frequenti è 3.

```
/* Es. 23.3 Meno frequente. Testo */
#include <iostream>
#include <math.h>
#include <stdlib.h>
#include <vector>
using namespace std;

int Conta(vector<int> v, int x)
 { /* Inserite qui la soluzione */ }
int MenoFrequente(vector<int> v)
 { /* Inserite qui la soluzione */ }

 int main(){
/* Definiamo un vettore C elencando gli elementi */
 int vettoreC[11] = {1, 2, 3, 2, 1, 1, 5, 2, 3, 2, 5};
/* Definiamo un vettore v del C++ a partire da vettoreC e la sua
lunghezza 11 */
 vector<int> v(vettoreC, vettoreC+11);
 cout << "MenoFrequente = " << MenoFrequente(v) << endl << "dovete
ottenere 3" << endl;
 system("PAUSE"); }
```

Es 23.4 (somma di una funzione su un segmento di vettore). Per questo esercizio è richiesta **una soluzione ricorsiva, altrimenti vale 0 punti.** Definite una funzione

double somma(vector<double> v, int a, int b, double f(double x))

che prenda in input un vettore v di reali e una funzione f dai reali ai reali. La funzione restituisce la somma

f(v[a]) + f(v[a+1]) + … + f(v[b])

dei valori di f in {v[a], …, v[b]}. **Esempio.** Se v = {-3, -2, -1, 0, +1, +2, +3} e f è la funzione **double** f(**double** x){**return** x*x;} allora somma(v,0,6,f) = 9+4+1+0+1+4+9 = 28 e somma(v,0,3,f) = 9+4+1+0 = 14. **Suggerimento.** Definite somma(v,a,b,f) per induzione sul numero di interi nel segmento [a,b].

```
/* Es. 23.4 Somma. Testo */
#include <iostream>
#include <math.h>
#include <stdlib.h>
#include <vector>
using namespace std;

double f(double x){return x*x;}

double somma(vector<double> v, int a, int b, double f(double x))
 { /* Inserite qui la soluzione */ }

int main(){
/* Definiamo un vettore C elencando gli elementi */
 double vettoreC[7] = {-3,-2,-1,0,+1,+2,+3};
/* Definiamo un vettore v del C++ a partire da vettoreC e la sua
lunghezza 7 */
 vector<double> v(vettoreC, vettoreC+7);
 cout << "somma(v,0,6,f) = " << somma(v,0,6,f) << endl;
 cout << " (deve fare 28) " << endl;
 cout << "somma(v,0,3,f) = " << somma(v,0,3,f) << endl;
 cout << " (deve fare 14) " << endl;
 system("PAUSE"); }
```

Lezione 23: Preparazione all'esame
Parte 2. Soluzioni

```
/* Es. 23.1. Minuscole. Soluzioni */
#include <iostream>
#include <math.h>
#include <stdlib.h>
using namespace std;

string stringToLower(string s)
{int i, len=s.length();
 for(i=0;i<len;i=i+1)
  {s[i] = tolower(s[i]);}
 return s;
 /*la stringa s restituita è solo una copia di s.
  La stringa s originale non viene modificata. */}

int main() {string s;
 cout << "Inserite una stringa con maiuscole:" <<endl;
 cin >> s;
 cout << "La stessa stringa in minuscole:" << endl <<
stringToLower(s) << endl;
 system("PAUSE"); }
```

```
/* Es. 23.2 La sommatoria L. Soluzioni. */
#include <iostream>
#include <math.h>
#include <stdlib.h>
using namespace std;

double L (double x, int n)
{double s;int i;
 for(i=1, s=0; i<=n; i=i+1)
  {s = s - pow(-x,i)/i; /* pow(-x,i) è un reale, dunque pow(-x,i)
è una divisione tra reali: */ }
 return s;
}

int main(){
 cout << "L(0.5,3)=" << L(0.5,3) << endl <<
  " dovrebbe valere circa 0.41667" << endl;
 cout << "L(0.5,10)=" << L(0.5,10) << endl
  << " dovrebbe valere circa 0.405435" << endl;
 system("PAUSE");}
```

```
/* Es. 23.3 Velore meno frequente. Soluzioni */
#include <iostream>
#include <math.h>
#include <stdlib.h>
#include <vector>
using namespace std;

int Conta(vector<int> v, int x)
{int i, len=v.size(); int count;
 for(i=0, count=0; i<len; i=i+1)
  {if (v[i]==x) {count=count+1;}; }
 return count;
}

int MenoFrequente(vector<int> v)
{int m=v[0], m_conta = Conta(v,m), len=v.size();
 for(int i=1; i<len; i++)
  {int v_conta = Conta(v,v[i]);
   if (v_conta < m_conta)
    {m=v[i]; m_conta = v_conta;}}
 return m;}


 int main(){ /* Definiamo un vettore C elencando gli elementi */
 int vettoreC[11] = {1, 2, 3, 2, 1, 1, 5, 2, 3, 2, 5};
/* Definiamo un vettore v del C++ a partire da vettoreC e dalla
sua lunghezza 11 */
 vector<int> v(vettoreC, vettoreC+11);
 cout << "MenoFrequente = " << MenoFrequente(v) << endl
  << "dovete ottenere 3" << endl;
 system("PAUSE"); }
```

```
/* Es. 23.4 Somma di un segmento di vettore. Soluzioni. */
#include <iostream>
#include <math.h>
#include <stdlib.h>
#include <vector>
using namespace std;

double f(double x){return x*x;}

/* definizione di somma(v,a,b,f)
   per induzione sul numero degli elementi in [a,b]*/
double somma(vector<double> v, int a, int b, double f(double x))
{if (a > b) /* somma segmento vuoto */
  return 0.;
 else /*a<=b*/
  return somma(v,a,b-1,f) + f(v[b]);
}

int main(){ /* Definiamo un vettore C elencando gli elementi */
 double vettoreC[7] = {-3,-2,-1,0,+1,+2,+3};
/* Definiamo un vettore v del C++ a partire da vettoreC e la sua
lunghezza 7 */
 vector<double> v(vettoreC, vettoreC+7);
 cout << "somma(v,0,6,f) = " << somma(v,0,6,f) << endl;
 cout << "somma(v,0,3,f) = " << somma(v,0,3,f) << endl;
 system("PAUSE"); }
```

Lezione 24: Preparazione all'esame
Parte 1. Esercizi

Anche in questa lezione assegniamo quattro esercizi di difficolta' simile a quelli che saranno dati all'esame del corso. Per ogni esercizio viene dato un **main** di prova che aiuta a controllare se la vostra soluzione è corretta. Per quattro esercizi all'esame vengono date due ore: qui diamo invece 90 minuti, e useremo gli ultimi 30 minuti della lezione per la correzione.

Gli esercizi di questa lezione riguardano stringhe, strutture, matrici e ricorsione.

Es. 24.1. Vettore delle Frequenze. Scrivete una funzione

vector<int> Frequenze(**string** s)

che data una stringa s restituisce un vettore di 26 interi, contenente al posto i la frequenza della lettera (maiuscola o minuscola) numero i dell'alfabeto inglese. **Suggerimento.** Se c è una maiuscola allora ***tolower(c)*** è la minuscola corrispondente, altrimenti tolower(c) = c. La posizione di una minuscola c nell'alfabeto inglese si calcola con la formula: ***pos = (c-'a')***. c è una minuscola ***se e solo se*** pos vale tra 0 e 25. Definite un vettore F di 26 interi tutto inizializzato a 0 per immagazzinare il risultato. Scorrete s con un ciclo, prendete la versione minuscola m del carattere trovato, e se m è una minuscola di posizione pos, aggiungete 1 a F[pos]. Alla fine restituite F.

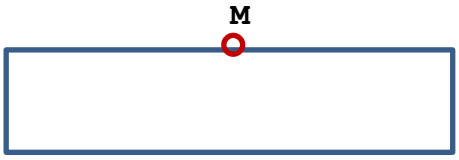
```
/* Es. 24.1 Vettore delle frequenze. Testo */
#include <iostream>
#include <math.h>
#include <stdlib.h>
#include <vector>
using namespace std;
vector<int> Vuoto(0); /* vettore vuoto */

vector<int> Frequenze(string s)
 { return Vuoto; /* Inserite la soluzione al posto di return
Vuoto; */ }

int main() {string s; cout << "Inserite una stringa: " << endl;
cin >> s; vector<int> F = Frequenze(s); int i;
for(i=0;i<26;i=i+1)
```

```
 {cout << "lettera " << (char) ('a'+i) << " : " << F[i] << endl;}
 system("PAUSE");
```

Es. 24.2 (Punto Medio Superiore). Scrivete una funzione

Point MedioSuperiore (Rectangle box)

che dato un rettangolo rappresentato dalla struttura descritta qui sotto e restituisce il punto medio M del bordo superiore del rettangolo, un elemento di tipo punto.



Esempio. Se il rettangolo ha margine inferiore sinistro {50,50} e base, altezza {200,300}, otteniamo come medio superiore M = {150,350}.

```
/* Es. 24.2 Medio Superiore. Testo */
#include <iostream>
#include <math.h>
#include <stdlib.h>
using namespace std;

struct Point {double x, y;}; //non dobbiamo dimenticare il ;

struct Rectangle
{Point corner;          //angolo inferiore sinistro
 double width, height;  //base, altezza
}; //non dobbiamo dimenticare il ;

void printPoint (Point p)
{cout << "(" << p.x << ", " << p.y << ")" << endl;}

//funzione che calcola il punto medio superiore di un rettangolo
Point MedioSuperiore (Rectangle box) {}

int main(){Point corner = { 50.0, 50.0 };
 Rectangle box  = { corner, 200.0, 300.0 };
 //margine inferiore sinistro = corner, base=200, altezza=300

 Point M = MedioSuperiore(box);
 cout << " punto medio superiore box = ";
 cout << "  (deve valere {150,350})" << endl;
 printPoint (M); system("PAUSE");}
```

Es 24.3 (Combinazione lineare). Definite una funzione
Matrix<double> comb(double a, Matrix<double> A, double b, Matrix<double> B)
che prenda in input due reali a, b e due matrici A, B di reali della stessa dimensione nxm. La funzione restituisce C = a*A + b*B. **Suggerimento.** Definite una nuova matrice C per contenere il risultato. Usate due cicli **for** annidati per assegnare ogni elemento di posto i,j di C.

```
/* Es. 24.3 Combinazione lineare. Testo */
#include <iostream>
#include <math.h>
#include <stdlib.h>
#include <vector>
#include "Matrix.cpp"
using namespace std;

Matrix<double> Vuota(0,0);
Matrix<double> comb(double a, Matrix<double> A,
double b, Matrix<double> B)
 { return Vuota; /* Rimpiazzate questa riga con la soluzione */ }

int main(){
/* Definiamo due vettori del linguaggio C con un elenco */
double vettoreA[9] = {1,2,3,
                      4,5,6,
                      7,8,9};
double vettoreB[9] = {1,1,1,
                      1,1,1,
                      1,1,1};
/* Definiamo due matrici C++ a partire da vettoreA, vettore B */
 Matrix<double> A(3,3,vettoreA);
 cout << endl << "A = " << A;
 Matrix<double> B(3,3,vettoreB);
 cout << endl << "B = " << B;
 cout << endl << "comb(1,A,10,B) = " << endl
  << comb(1,A,10,B) << endl;
 system("PAUSE"); }
```

Es 24.4 (somma dei valori positivi di un segmento di vettore). Per questo esercizio è richiesta una soluzione ricorsiva. Definite una funzione

double somma(vector<double> v, int a, int b)

che prende un vettore v di reali, due indici a,b del vettore, e restituisce la somma degli elementi positivi di v con indice in [a,b]. **Esempio.** Se v = {-3, -2, -1, 0, +1, +2, +3} allora somma(v,0,6) = 1+2+3 = 6 e somma(v,0,3) = 0 (non ci sono elementi positivi con indice in [0,3]. **Suggerimento.** Definite la funzione somma(v,a,b) per induzione sul numero di interi nel segmento [a,b].

```
/* Es. 24.4 Estrazione di un segmento. Testo */
#include <iostream>
#include <math.h>
#include <stdlib.h>
#include <vector>
using namespace std;

void printVett(vector<double> V)
 {int i, len=V.size();
  for(i=0;i<len;i=i+1){cout << V[i] << " ";};
  cout << endl;}

double somma(vector<double> v, int a, int b)
 { /* Inserite qui la soluzione */ }

int main(){/* Definiamo un vettore C elencando gli elementi */
 double vettoreC[7] = {-3,-2,-1,0,+1,+2,+3}; /* Definiamo un
vettore v del C++ a partire da vettoreC e la sua lunghezza 7 */
 vector<double> v(vettoreC, vettoreC+7);
 cout << "v             = "; printVett(v);
 cout << "somma(v,0,6) = " << somma(v,0,6) << endl;
 cout << "somma(v,0,3) = " << somma(v,0,3) << endl;
 system("PAUSE"); }
```

Lezione 24: Preparazione all'esame
Parte 2. Soluzioni

```
/* Es. 24.1 Frequenze delle lettere. Soluzioni */
#include <iostream>
#include <math.h>
#include <stdlib.h>
#include <vector>
using namespace std;
vector<int> Vuoto(0);

vector<int> Frequenze(string s)
{vector<int> F(26,0); int i, len = s.length();
for(i=0;i<len;i=i+1)
  {char m = tolower(s[i]); int pos=(m-'a');
   if ((0<=pos) && (pos<=25))
    {F[pos]=F[pos]+1;}}
 return F;}

int main() {string s; cout << "Inserite una stringa: " << endl;
cin >> s;
vector<int> F = Frequenze(s); int i;
for(i=0;i<26;i=i+1)
  cout << "lettera " << (char) ('a'+i) << " : " << F[i] << endl;
 system("PAUSE"); }
```

```
/* Es. 24.2 Punto Medio Superiore. Soluzioni */
#include <iostream>
#include <math.h>
#include <stdlib.h>
using namespace std;

struct Point
 {double x, y;}; //non dobbiamo dimenticare il ;

struct Rectangle
{
 Point corner;          //angolo inferiore sinistro
 double width, height;  //base, altezza
};  //non dobbiamo dimenticare il ;

void printPoint (Point p)
{cout << "(" << p.x << ", " << p.y << ")" << endl;}

//funzione che calcola il punto medio superiore di un rettangolo
Point MedioSuperiore (Rectangle box)
{
 double x = box.corner.x + box.width/2;
 double y = box.corner.y + box.height;
 Point result = {x, y};
 return result;
}

int main(){Point corner = { 50.0, 50.0 };
 Rectangle box  = { corner, 200.0, 300.0 };
 //margine inferiore sinistro = corner, base=200, altezza=300

 Point M = MedioSuperiore(box);
 cout << " punto medio superiore box = ";
 printPoint (M); system("PAUSE");}
```

```
/* Es. 24.3 Combinazione lineare di Matrici. Soluzioni */
#include <iostream>
#include <math.h>
#include <stdlib.h>
#include <vector>
#include "Matrix.cpp"
using namespace std;

Matrix<double> Vuota(0,0);

Matrix<double> comb(double a, Matrix<double> A,
double b, Matrix<double> B)
{int i,j, n = A.Row(), m = A.Col();
 Matrix<double> C(n,m);
 for(i=0;i<n;i=i+1)
  for(j=0;j<m;j=j+1)
   {C[i][j] = a*A[i][j] + b*B[i][j];}
 return C;
}


int main(){ /* Definiamo due vettori del linguaggio C elencandone
gli elementi */
double vettoreA[9] = {1,2,3,
                      4,5,6,
                      7,8,9};
double vettoreB[9] = {1,1,1,
                      1,1,1,
                      1,1,1};
/* Definiamo due matrici del C++ a partire da vettoreA, vettoreB*/
 Matrix<double> A(3,3,vettoreA);
 cout << endl << "A = " << endl << A;
 Matrix<double> B(3,3,vettoreB);
 cout << endl << "B = " << endl << B;
 cout << endl << "comb(1,A,10,B) = " << endl << comb(1,A,10,B) <<
endl;
 system("PAUSE"); }
```

```
/* Es. 24.4 Estrazione di un segmento. Soluzioni */
#include <iostream>
#include <math.h>
#include <stdlib.h>
#include <vector>
using namespace std;

void printVett(vector<double> V)
 {int i, len=V.size();
  for(i=0;i<len;i=i+1){cout << V[i] << " ";};
  cout << endl;}

double somma(vector<double> v, int a, int b)
 { if (a>b)                    return 0;
   else /* a<=b */ if (v[a]>0) return v[a]+somma(v,a+1,b);
   else /* a<=b e v[a]<=0*/    return somma(v,a+1,b);
 }

int main(){/* Definiamo un vettore C elencando gli elementi */
 double vettoreC[7] = {-3,-2,-1,0,+1,+2,+3}; /* Definiamo un
vettore v del C++ a partire da vettoreC e la sua lunghezza 7 */
 vector<double> v(vettoreC, vettoreC+7);
 cout << "v             = "; printVett(v);
 cout << "somma(v,0,6) = " << somma(v,0,6) << endl;
 cout << "somma(v,0,3) = " << somma(v,0,3) << endl;
 system("PAUSE"); }
```

Tutorato 12: Matrici (soluzioni di sistemi) Parte 1. Esercizi proposti

In questo tutorato si chiede di completare un **main di prova** (qui sotto), come all'esame. Le funzioni da completare sono elencate di seguito. Tra le altre, viene richiesta una funzione che **risolve un sistema**, come nella Lezione 22. Questa volta si richiede di definire tale funzione utilizzando la matrice inversa.

Tut12.1 Scrivere una funzione
Matrix<**double**> ruotaSx(**Matrix**<**double**> A)
che restituisce la matrice corrispondente alla rotazione di 90 gradi in senso antiorario di A.

Tut12.2 Scrivere una funzione
Matrix<**double**> ruotaDx(**Matrix**<**double**> A)
che restituisce la matrice corrispondente alla rotazione di 90 gradi in senso orario di A.

Tut12.3 Scrivere una funzione
vector<**double**> prodotto(**Matrix**<**double**> A, **vector**<**double**> v)
che restituisce il vettore prodotto matrice per vettore A*v.

Tut12.4 Scrivere una funzione
Matrix<**double**> Inversa(**Matrix**<**double**> A)
Che restituisce la matrice inversa di A. Usate le funzioni ***minor, det*** definite nella Lezione 22, e la funzione ***trasposta*** definita in Tut11.3, Usate le formule:

$$A^{-1} = \frac{1}{\det(A)} \cdot (\operatorname{cof} A)^T$$

dove $\mathbf{B}^{\mathbf{T}}$ indica la trasposta di B, e (cof A) è definita come segue:

$$\operatorname{cof}_{i,j}(A) := (-1)^{i+j} \cdot \det(A_{i,j})$$

Tut12.5 Scrivere una funzione
vector<double> Sistema (**Matrix<double>** A, **vector<double>** C)
che calcola il vettore delle soluzioni del sistema lineare AX=C come ***X=prodotto(Inversa(A,C).*** Usate le funzioni prodotto e Inversa degli esercizi 3 e 4.

Utilizzate come **main** di prova il seguente:

```
int main()
{
    double a[] = { -1,  3,  5, 6,
                    1,  0,  0, 18,
                    1,  2,  -2, 0};
```

```
    Matrix<double> A(3, 4, a);
    cout << "La matrice A: " << endl << A;
    cout << "Tut12.1: rotazione a sinistra (antioraria) di 90
gradi " << endl << ruotaSx(A);
    cout << "Tut12.2: rotazione a destra (oraria) di 90 gradi "
     << endl << ruotaDx(A);
    system("pause"); system("cls");

    cout << "Tut 12.3,4,5. Dato il sistema:" << endl <<
     "x  -y +z = +6" << endl <<
     "2x +y -z = -3" << endl <<
     "x  -y -z =  0" << endl <<
     "ne calcoliamo le soluzioni col metodo dell'inversa:" << endl;

     // S matrice dei coefficienti
     double s[] =
     {    1,  -1,  1,
          2,   1, -1,
          1,  -1, -1  };

     Matrix<double> S(3, 3, s);
     cout << endl << "Matrice dei coefficienti:" << endl;
     cout << S << endl;

     // V è il vettore colonna dei termini noti
     vector<double> V(3);
     V[0] = 6;
     V[1] = -3;
     V[2] = 0;
     cout << "vettore dei termini noti" << endl;
     for(int i = 0; i < 3; i++)
          cout << V[i] << endl;

     // X è il vettore delle soluzioni, se esistono
     vector<double> X = SistemaInversa(S, V);
     if(X.size() == 0)
          cout << "Non ci sono soluzioni" << endl;
     for(int i = 0; i < X.size(); i++)
          cout << "Soluzione x_" << i+1 << " = " << X[i] << endl;
     cout << endl << endl <<
    "l'output dovrebbe essere:" << endl <<
     "Soluzione x_1 = 1"        << endl <<
     "Soluzione x_2 = -2"       << endl <<
     "Soluzione x_3 = 3"        << endl;
    system("pause");system("cls");}
```

Tutorato 12: Matrici (soluzioni di sistemi)
Parte 2. Soluzioni

```
#include <iostream>
#include <stdlib.h>
#include <vector>
#include <math.h>
#include "Matrix.cpp"
using namespace std;

/* Minore algebrico: ricopiata dalla Lezione 22 */
Matrix<double> minor(Matrix<double> M, int i, int j)
// pre:  M è una matrice nxn e 0 <=i,j < n
// post: calcola il minore M^(i,j) ossia
//       M meno la riga i e la colonna j
{   Matrix<double> A(M.Row()-1, M.Col()-1);
    // copia le parti rilevanti di M in A
    for (int h = 0; h < A.Row(); h++)
          for(int k = 0; k < A.Col(); k++)
               if (h < i && k < j) A[h][k] = M[h][k];
               else if (h < i && k >= j) A[h][k] = M[h][k+1];
               else if (h >= i && k < j) A[h][k] = M[h+1][k];
               else // h >= i && k >= j
                    A[h][k] = M[h+1][k+1];
    return A;
}

/* Determinante: ricopiata dalla Lezione 22 */
double det(Matrix<double> M)
// pre:  M è una matrice quadrata
// post: calcola il determinante di M
{   if (M.Row() == 2)
          return M[0][0]*M[1][1] - M[0][1]*M[1][0];
     else
     {   double d = 0.0;
          for(int j = 0; j < M.Col(); j++)
          {   Matrix<double> A = minor(M, 0, j);
               double m = det(A);
               if (j % 2 == 0) d += M[0][j] * m;
               else d -= M[0][j] * m;
          }
          return d;
     }
}
```

```
/* Tut12.1 rotazione a sinistra */
Matrix<double> ruotaSx(Matrix<double> A)
{
      Matrix<double> B(A.Col(), A.Row());
      for (int i = 0; i < B.Row(); i++)
           for (int j = 0; j < B.Col(); j++)
               B[i][j] = A[j][A.Col()-1-i];
      return B;
}

/* Tut12.2 rotazione a destra */
Matrix<double> ruotaDx(Matrix<double> A)
{
      Matrix<double> B(A.Col(), A.Row());
      for (int i = 0; i < B.Row(); i++)
           for (int j = 0; j < B.Col(); j++)
               B[i][j] = A[A.Row()-1-j][i];
      return B;
}

/* Tut12.3 prodotto matrice per vettore */
vector<double> prodotto(Matrix<double> A, vector<double> v)
// pre: A.Col() == B.Row()
// post: ritorna il prodotto A x B righe per colonne
{   double s;
    vector<double> w (A.Row());
    for (int i = 0; i < A.Row(); i++)
        { s=0.0;
             for (int j = 0; j < v.size(); j++)
                s = s + A[i][j]*v[j];
         w[i] = s;
        }
    return w;
}

/* Trasposta */
Matrix<double> trasposta(Matrix<double> M)
{
    Matrix<double> T(M.Col(), M.Row());
    for (int i = 0; i < T.Row(); i++)
        for (int j = 0; j < T.Col(); j++)
            T[i][j] = M[j][i];
    return T;
}
```

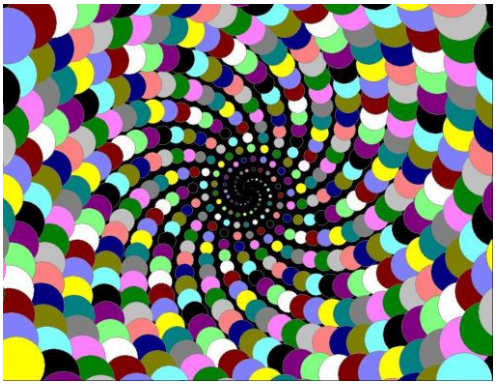
```
/* Tut12.4 Matrice inversa */
Matrix<double> Inversa(Matrix<double> A)
{
 double det_A = det(A);
 Matrix<double>Inv (A.Row(), A.Col());
 for(int i=0; i< Inv.Row(); i++)
  for(int j=0; j< Inv.Col(); j++)
   {Inv[i][j]=(pow(-1.0, i+j)* det(minor(A,i,j)))/det_A;}
 return trasposta(Inv);
}

/* Tut12.5 Risoluzione di sistema */
vector<double> SistemaInversa(Matrix<double> A, vector<double> C)
// pre:  A matrice quadrata nxn, C vettore di dim. n
// post: se A è non singolare ritorna il vettore X tale che AX = C
//       altrimenti ritorna il vettore "vuoto" (di dimensione 0)
{
     double det_A = det(A);
     if(det_A == 0) // A è singolare
     {
          vector<double> X;
          return X;
     }
     else // A è non singolare
     {
          return prodotto(Inversa(A), C);
     }
}


int main()
{
    double a[] = { -1,  3,  5, 6,
                    1,  0,  0, 18,
                    1,  2,  -2, 0};
    Matrix<double> A(3, 4, a);
    cout << "La matrice A: " << endl << A;
    cout << "Tut12.1: rotazione a sinistra (antioraria) di 90
gradi " << endl << ruotaSx(A);
    cout << "Tut12.2: rotazione a destra (oraria) di 90 gradi "
     << endl << ruotaDx(A);
    system("pause"); system("cls");

    cout << "Tut 12.3,4,5. Dato il sistema:" << endl <<
     "x  -y +z = +6" << endl <<
```

```
    "2x +y -z = -3" << endl <<
    "x  -y -z =  0" << endl <<

    "ne calcoliamo le soluzioni col metodo dell'inversa:" << endl;

    // S matrice dei coefficienti
    double s[] =
    {    1,  -1,  1,
         2,   1, -1,
         1,  -1, -1  };

    Matrix<double> S(3, 3, s);
    cout << endl << "Matrice dei coefficienti:" << endl;
    cout << S << endl;

    // V è il vettore colonna dei termini noti
    vector<double> V(3);
    V[0] = 6;
    V[1] = -3;
    V[2] = 0;
    cout << "vettore dei termini noti" << endl;
    for(int i = 0; i < 3; i++)
         cout << V[i] << endl;

    // X è il vettore delle soluzioni, se esistono

    vector<double> X = SistemaInversa(S, V);
    if(X.size() == 0)
         cout << "Non ci sono soluzioni" << endl;
    for(int i = 0; i < X.size(); i++)
         cout << "Soluzione x_" << i+1 << " = " << X[i] << endl;
    cout << endl << endl <<
    "l'output dovrebbe essere:" << endl <<
    "Soluzione x_1 = 1"         << endl <<
    "Soluzione x_2 = -2"        << endl <<
    "Soluzione x_3 = 3"         << endl;
   system("pause");system("cls");
}
```

Fine del Corso:
Basi di Informatica 2016/2017
Corso di Laurea in Matematica
Università di Torino

Appendice al Diario delle Lezioni Di Basi di Informatica del 2016/2017

1. Appendice 1. Un esempio di programma: una animazione di spirali (non fa parte del corso).
2. Tutorato 13 (di solito non viene svolto nel corso): esercizi di preparazione all'esame.



Spirale di Archimede disegnata con cerchi di 16 colori

Appendice 1.
Un esempio di programma: una animazione di spirali
(non fa parte del corso)

Nel corso di Basi di Informatica, per ragioni di tempo, non abbiamo spiegato come usare la libreria di grafica. Tuttavia il seguente programma di grafica puo' venire letto e compreso nelle sue linee generali anche ***senza*** conoscere la libreria di grafica, ed costituisce un buon esempio di come si progetta un programma. Mostriamo come definire:

1. una funzione che disegna un singolo cerchio
2. a partire da essa, una seconda funzione che disegna 600 cerchi disposti lungo una spirale,
3. a partire da quest'ultima, una terza funzione che disegna 1500 spirali leggermente diverse una dall'altra (dunque 600 x 1500 = 900 mila cerchi), creando un'illusione di animazione delle spirali.

Non potete compilare il programma senza prima scaricare e istallare la libreria di grafica. Quindi, se volete provare come funziona, vi consigliamo di scaricare direttamente l'eseguibile:
http://www.di.unito.it/~stefano/C-2010-Graphics02-AnimazioneSpirali.exe

(Per chi fosse curioso, spieghiamo come scaricare, istallare e usare la libreria di grafica nel sito del corso di Programmazione Avanzata del 2014/2015:
http://math.i-learn.unito.it/mod/page/view.php?id=26870
Ripetiamo un'ultima volta che questa libreria ***non fa parte del corso di Basi di Informatica****.)*

```
#include <graphics.h>
#include <iostream>
#include <math.h>
using namespace std;

/* A SEQUENCE OF BALLS ALONG AN ARCHIMEDIAN SPIRAL
We will draw a sequence of balls with increasing radius, whose
centers are in an archimedian spiral of parametric equation:
 x(t) = x0 + t*cos(alpha*t)
 y(t) = y0 + t*sin(alpha*t)
where
 alpha = angular speed
This archimedian spiral is the path of a point rotating of an
angle alpha per unit of time, and moving away from the origin with
speed 1. The ball number t has center (x(t),y(t)) and radius
Radius*t.
```

```
We produce an animation, that is, a sequence of drawing we call
"frames". The difference between two consecutive  frames is the
value of the angle alpha: we let alpha grow very slightly between
two consecutive frames. We paint the ball cyclically of 16 basic
colors.

BASE AND HEIGHT of the graphic window. */
int N=1000, M=768;
/* The y component is downward-directed:

Origin           x=N
   *----------------*
   |                |
   |                |
y=M|                |
   *----------------*

NOTE 1: the next command of the form:
         fillellipse(x,y,Radiusx,Radiusy)
will draw an ellipse with the colors we selected for the border
and for the inside, and with center (x,y). The two radius of the
ellipse must be equal if we want to draw a cycle.

NOTE 2: the next command of the form:
          setfillstyle(PATTERN,C);
sets the color filling a figure color to the given color C and
sets the "pattern" (the way a color is painted). The pattern
"SOLID_FILL" completely fills the figure with the color, in any
other pattern the background color is visible. */

double Radius = 0.1; //radius of the first ball

/*1. THE FUNCTION BALL draws the ball number t of a spiral with
parameter alpha */
void ball(double alpha, int t)
{ //we choose 16 colors cyclically in the parameter t
  setfillstyle(SOLID_FILL,t%16);
  fillellipse(
  (int) (N/2+t*sin(t*alpha)),
  (int) (M/2+t*cos(t*alpha)),
  (int) (t*Radius),
  (int) (t*Radius));
}

/*2. THE FUNCTION SPIRAL draws a spiral with parameters alpha,
Radius and a number of balls Nballs */
int    NBalls = 600; //number of balls of each spiral

void spiral(double alpha)
{int t=1;
 while(t<=NBalls){ball(alpha,t);t=t+1;}
}
```

```
/* THE FUNCTION ANIMATESPIRAL draws an animation with many
slightly different spirals. Spirals are indexed from frame1 to
frame2, have the parameter alpha of values:
  alpha=frame1*e,(frame1+1)*e,(frame1+2)*e, ...
Spirals have the same Radius for balls and the same numbers NBalls
of balls.*/
//animation frames are numbered from frame1 to frame2
 int frame1=1, frame2 = 1500;
//increment for alpha between two consecutive frames
 double e = 0.0003;

void animatespiral()
{swapbuffers();
/* makes invisible everything we draw until the next command
"swapbuffers();". It is used to speed up drawing */
 cleardevice();
/* initializes the window for the next drawing filling it with the
"background" color and erasing everything else */
 double alpha;
 alpha = frame1*e; //we initialize alpha
 while (alpha <= frame2*e)
  {spiral(alpha);  //draw the current spiral
   swapbuffers();  //makes the drawing visible
   delay(20);      //waits for 20 milliseconds
                   //(otherwise the animation runs too fast)
   cleardevice();  //initializes the internal memory
                   //for the next spiral (if any)
   alpha=alpha+e;  //we increment alpha for the next spiral
  }
}


main(){
///////////////BEGIN INITIALIZATION //////////
 initwindow(N,M);   //opens a NxM pixels graphics window.
///////////////END INITIALIZATION//////////

/* We have 16 Basic Colors available: BLACK, BLUE, GREEN, CYAN,
RED, MAGENTA, BROWN, LIGHTGRAY, DARKGRAY, LIGHTBLUE, LIGHTGREEN,
LIGHTCYAN, LIGHTRED, LIGHTMAGENTA, YELLOW, WHITE */

setbkcolor(BLACK); /* setbckcolor(C) sets the background color for
screen (it does not draw the color!) */

/* More colors: we may define: int C = Color(red,green,blue);
where red, green, blue are integers from 0 to 255, expressing the
quantity of red, green, blue in the color C */

int GRAY = COLOR(127,127,127); /* when the quantities of red,
green, blue are the same we obtain some degree of gray:(0,0,0) is
black and (255,255,255) is white.*/
```

```
setcolor(GRAY); /* the command setcolor(C) sets the color of the
border line of any future figure to the given color C */

/* 1. We draw the ball number i of the spiral with parameter
alpha, for some alpha, i. */
 double alpha  = 1.5; //one possible choice for the angular speed
 int i = NBalls/2;    //one possible cohordinate for the ball
 ball(alpha,i);
 delay(3000);         //we wait for that many milliseconds

/* 2. We draw the spiral with parameter alpha, for some alpha */
 spiral(alpha);
 delay(3000);         //we wait for that many milliseconds

/* 3. We draw the animation. All parameters: the number of the
first and last frame, the increment e, the number NBalls of balls
have already been decided. */
 animatespiral();
 system("pause");}
```

Tutorato 13: preparazione all'esame
Parte 1. Esercizi proposti

A seconda delle circostanze del corso, questo tutorato viene effettivamente svolto con una lezione in più, oppure no. In ogni caso consigliamo di svolgere questi esercizi come preparazione all'esame.

Tut13.1. Un suffisso non vuoto è la parte di una stringa da un certo punto i poi. I suffissi non vuoti di "mare" sono: "mare", "are", "re", "e". Definite una funzione:

```
void Suffissi(string s)
```

che, presa in input una stringa, stampi a video tutti i suoi suffissi non vuoti. **Suggerimento.** Definite prima una funzione **void** stampa(**string** s, **int** i) che stampa il suffisso di s a partire dal carattere di posto i.

Tut13.2 Scrivere una funzione

```
int PrimoRipetuto(vector<int> V)
```

che prenda come argomento un vettore V, e restituisca la ***posizione*** del primo valore V[i] uguale a V[i+1] (al valore seguente in V) se esiste, e -1 se non ce ne sono. Nel caso ci siano più posizioni possibili perché coppie di valori consecutivi uguali compaiono più volte nel vettore, si chiede di restituire la prima da sinistra. **Attenzione.** La funzione non deve restituire il valore uguale al valore seguente, ma solo la sua ***posizione***. **Esempi.** Se V1={5,12,5}, allora PrimoRipetuto(V1,3) = -1 (i due valori 5 non sono consecutivi), mentre se V1={5,5,5} allora PrimoRipetuto(V1,3) = 0, la posizione del primo valore di V uguale a un valore consecutivo.

Tut13.3. Data una matrice quadrata A, scrivere una funzione

```
void SommaInDiagonale(Matrix<double> &M)
```

che la modifica inserendo al posto di ogni elemento della diagonale la somma dei valori della riga (comprendendo nella somma anche il l'elemento stesso). **Esempio.** Se A = {{1,2},{3,4}}, allora A modificata diventa {{1+2,2}, {3,3+4}}.

Tut13.4 Scrivere una funzione ***ricorsiva***

```
int funT(int n)
```

per il calcolo dei numeri T(n) definiti induttivamente come segue: T(0)=0 ; T(1)=1; T(n)= 2*T(n-2) + T(n-1) se n>1. **Esempio.** T(10) = 341.

Utilizzate come **main di prova** per tutti gli esercizi il seguente:

```
#include <iostream>
#include <stdlib.h>
#include <vector>
#include <math.h>
```

```
#include "Matrix.cpp"
using namespace std;

void Suffissi(string s){}
int PrimoRipetuto(vector<int> V){}
void SommaInDiagonale(Matrix<double> &M){}
int funT(int n){}

int main()
{string stringa = "provapersuffissi";
 Suffissi(stringa);
 system("pause"); system("cls");

 int n=3;
 vector<int> V(3); V[0] = 2; V[1] = 12; V[2] = 5;
 cout << "V = {2,12,5}" << endl;
 cout << "PrimoRipetuto(V) =" << PrimoRipetuto(V) << endl;
 cout << "Il risultato deve essere -1" << endl << endl;
 V[1]=5; // Adesso V={2,5,5}
 cout << "V = {2,5,5}" << endl;
 cout << "PrimoRipetuto(V) =" << PrimoRipetuto(V) << endl;
 cout << "Il risultato deve essere 1" << endl << endl;
 V[0]=5; // Adesso V={5,5,5}
 cout << "V = {5,5,5}" << endl;
 cout << "PrimoRipetuto(V) =" << PrimoRipetuto(V) << endl;
 cout << "Il risultato deve essere 0" << endl <<endl;
 system("pause"); system("cls");

 double a[] = { 8 , 14, 6,
                 1,  9, 18,
                20, 10, -2 };
 Matrix<double> A(3, 3, a);
 cout << "La matrice A: " << endl << A;
 SommaInDiagonale(A);
 cout << "SommaInDiagonale(A)"<< endl<< A << endl;
 cout << "La diagonale deve essere {28,28,28}" << endl <<endl;
 system("pause");system("cls");

 cout << "funT(3)=" << funT(3) << endl;
 cout << "Il risultato deve essere 3" << endl <<endl;
 cout <<"funT(5)=" << funT(5) << endl;
 cout << "Il risultato deve essere 11" << endl <<endl;
 cout <<"funT(7)=" << funT(7) << endl;
 cout << "Il risultato deve essere 43" << endl <<endl;
 cout <<"funT(10)=" << funT(10) << endl;
 cout << "Il risultato deve essere 341" << endl <<endl;
 system("pause"); system("cls");}
```

Tutorato 13: preparazione all'esame
Parte 2. Soluzioni

Tut13.1 Si realizzi un programma

```
void Suffissi(string s)
```

che, presa in input una stringa, stampi a video tutti i suoi suffissi.

```
void stampa(string s, int i)
{for (int j=i; j<s.length(); j++){cout<<s[j];} cout << endl;}

void Suffissi(string s)
{
     int i=0;
     while( i < l )
     {
          stampa(s,i);
          i++;
     }
}
```

Tut13.2 Scrivere una funzione

```
int PrimoRipetuto(vector<int> V)
```

che prenda come argomento un vettore V, e restituisca la posizione del primo valore in V1[0 .. n-1] uguale al valore seguente, -1 se non ce ne sono. La funzione non deve restituire il valore uguale al valore seguente, ma solo la sua posizione. Nel caso ci siano più posizioni possibili perché lo stessa coppia di valori consecutivi uguali compare più volte nel vettore, si chiede di restituire la prima da sinistra. **Esempi.** Se V1={5,12,5}, allora PrimoRipetuto(V1,3) = -1 (i due valori 5 non sono consecutivi), mentre se V1={5,5,5} allora PrimoRipetuto(V1,3)=0, la posizione del primo valore di V uguale a un valore consecutivo.

```
int PrimoRipetuto(vector<int> V)
 {
     int i;
     for (i=0;i<=V.size()-2;++i)
           if (V[i]==V[i+1])
                 return i;
      return(-1);
 }
```

Tut13.3 Data una matrice quadrata, scrivere una funzione che la modifica inserendo sulla diagonale la somma dei valori della riga (comprendendo nella somma anche il valore sulla diagonale).

```
void SommaInDiagonale(Matrix<double> &M)
{
    for(int i=0; i< M.Row(); i++ )
         for(int j=0; j < M.Col(); j++ )
              {
                   if(j!=i)  M[i][i]+=M[i][j];
              }
}
```

Tut13.4 Scrivere una funzione ricorsiva

```
int funT(int n)
```

per il calcolo dei numeri T(n) definiti dalle seguenti relazioni: T(0)=0 ; T(1)=1; T(n)= 2*T(n-2) + T(n-1) se n>1.

```
int funT(int n)
 {
     if (n==0 || n==1 ) return n;
     else return 2*funT(n-2) + funT(n-1);

 }
```

Fine dell'Appendice al Corso di Basi di Informatica del 2016/2017