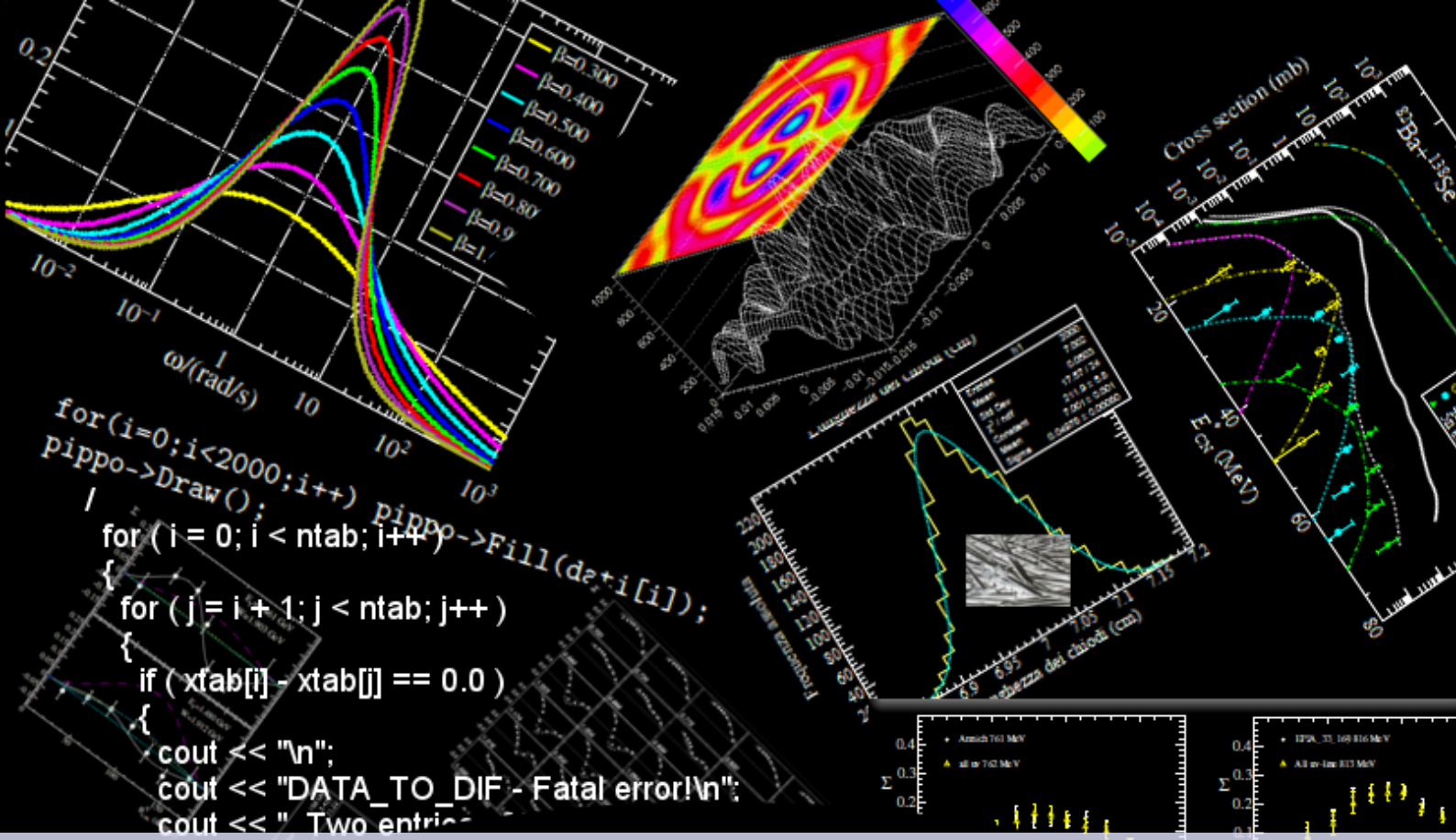
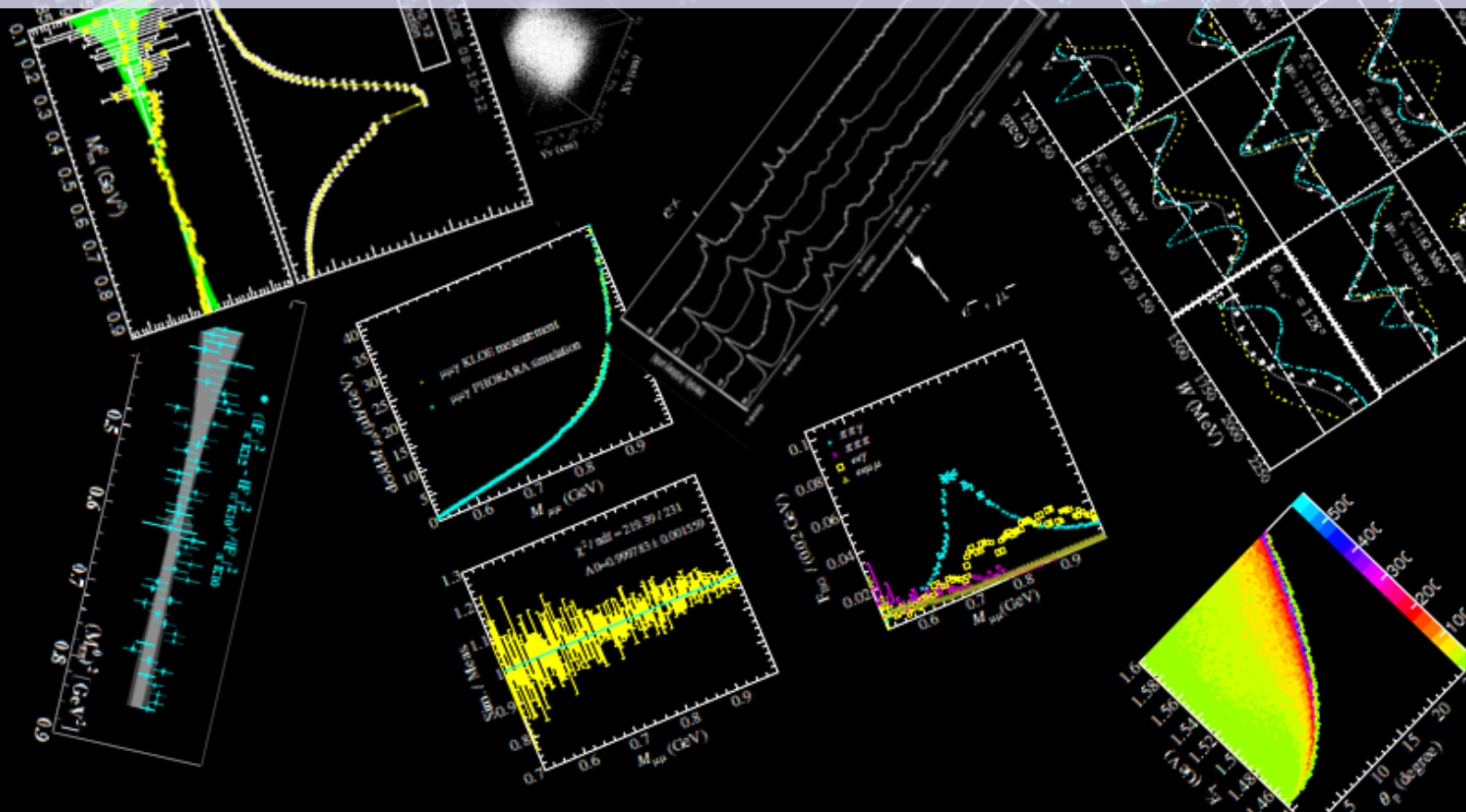


Laboratorio Informatico

Appunti al corso anno accademico 2016/17

Giuseppe Mandaglio



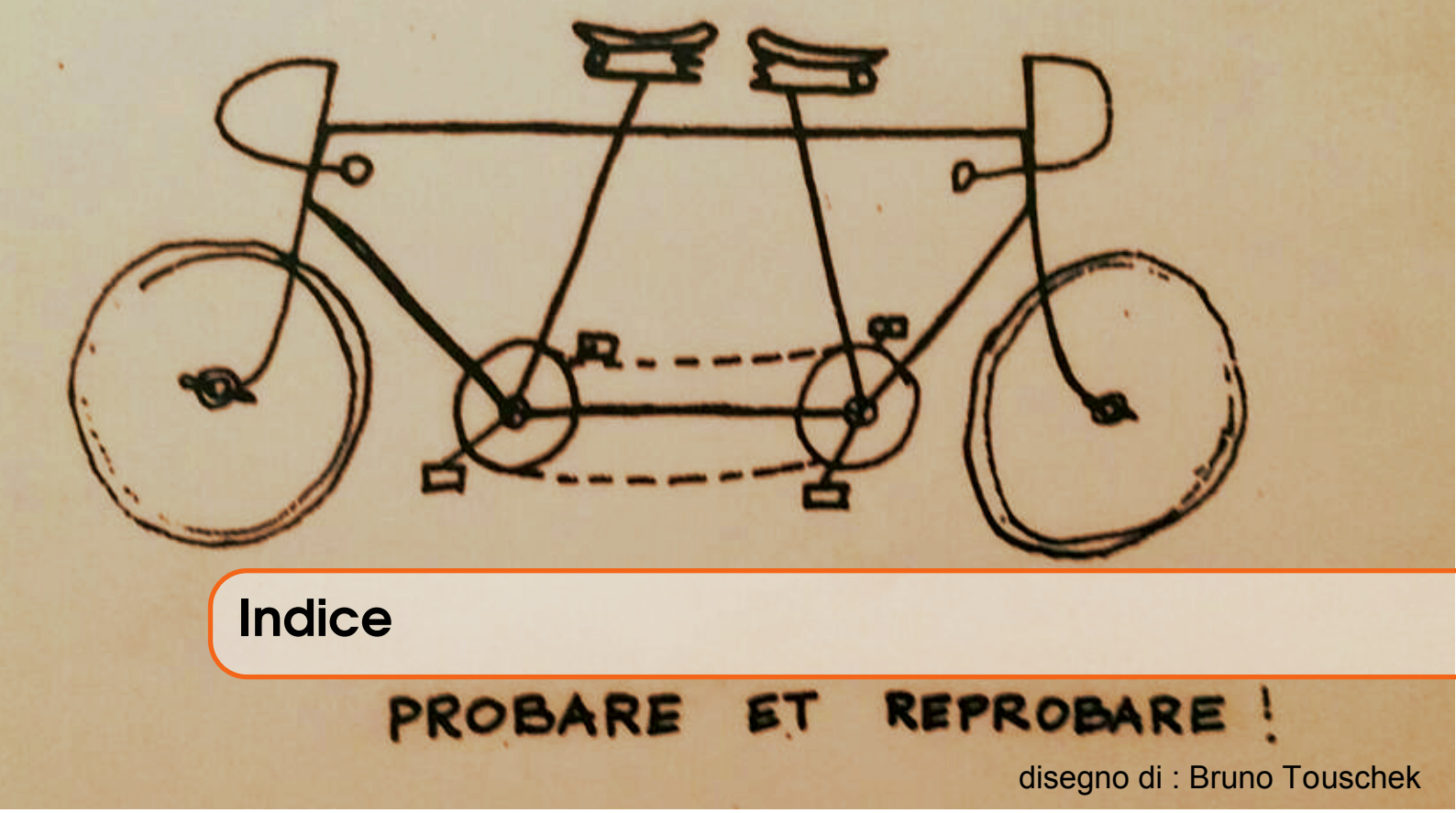
PUBLISHED BY CDL FISICA - UNIV. MESSINA

BOOK-WEBSITE.COM

First printing, ??

Prefazione

Il corso di Laboratorio Informatico per il corso di Laurea Triennale in Fisica è stato ideato dal Consiglio di Corso di Laurea come un insegnamento che fornisse i primi rudimenti alla programmazione e all'analisi dei dati. Il corso di Laurea in Fisica è in assoluto il percorso di studi che contempla il maggior numero di insegnamenti di Laboratorio tra tutti i corsi di laurea nei vari atenei, e questo è dovuto alla natura sperimentale di questa disciplina. Quindi, questo corso deve necessariamente fornire agli studenti gli strumenti informatici di base per analizzare i dati che vengono raccolti negli esperimenti condotti nei vari corsi di laboratorio.
Il corso inoltre deve rendere gli studenti appena iscritti al corso di laurea e provenienti da diversi scuole capaci di tradurre i metodi di risoluzione dei problemi in algoritmi interpretabili ed eseguibili da un calcolatore. Queste abilità sono fondamentali per affrontare al meglio gli altri corsi e propedeutiche ai corsi di Fisica Computazione e di Analisi Dati che vengono tenuti nel corso di Laurea Magistrale in Fisica dell'Università di Messina.
Il linguaggio di programmazione che sarà utilizzato nel corso è il C++ standard. I motivi sono molteplici, questo linguaggio di programmazione può essere utilizzato sia in programmi ad alto-livello che a basso-livello (alto e basso si riferiscono a quale livello il linguaggio riesce a interagire con la macchina)-
Verrà inoltre presentato un parallelo tra la programmazione procedurale in C++, che equivale alla programmazione in C, e quella in Fortran in modo da consentire agli studenti di comprendere un codice scritto in questo linguaggio, modificarlo o re-implementarlo. Il Fortran è un linguaggio di programmazione molto potente per il calcolo scientifico ed ancora oggi esiste una varietà infinità di programmi scritti in questo linguaggio principalmente da Fisici.
Il corso è rivolto a futuri Fisici, quindi i metodi numerici e la formulazione di algoritmi originali saranno privilegiati rispetto alle spiegazioni meticolose sulle infinite possibilità offerte del dato linguaggio di programmazione. In altre parole, verrà dato maggiore riguardo alla programmazione che al linguaggio, e di quest'ultimo sarà trasmesso giusto l'essenziale necessario per implementare i propri algoritmi.
Un terzo delle ore di questo corso sarà riservato all'apprendimento dei primi rudimenti del framework di analisi dati ROOT (software libero rilasciato dal CERN interamente governabile con il C++) e di un programma leggero e potente come gnuplot, che forniranno agli studenti strumenti potenti per l'analisi e la rappresentazione grafica dei dati che raccoglieranno nel corso di Laurea e nella loro futura vita di professionisti della scienza. Root è una delle piattaforme software più potenti e utilizzate nel campo della Fisica e non solo, mentre gnuplot è un programma leggero e potente per la grafica e l'analisi elementare dei dati capace di funzionare su qualunque dispositivo.
Il sistema operativo di riferimento del corso è Linux. Nel presente corso sarà utilizzato solo software libero, essenzialmente codici sorgente, librerie e compilatori. Linux è l'ambiente ideale dove trovare tutto quello che serve al riparo da virus e dalla dipendenza da programmi commerciali i cui codici sorgenti sono inaccessibili.
In questa raccolta di appunti, ci sono numerosi esempi e i codici implementati durante il corso. Nei codici ci sono numerosi commenti che devono essere considerati parte integrante e fondamentale di questo testo.



disegno di : Bruno Touschek

Indice

I **Linux**

1 **Comandi base** 9

1.1 **Shell-iniziare a muoversi sulla riga di comando** 9

1.2 **Compilatori** 11

II **Programmazione in C++**

2 **Introduzione** 15

2.1 **Stuttura di un programma** 16

2.2 **Operazioni di input-output I** 18

3 **Elementi base per la programmazione** 21

3.1 **Variabili I** 21

3.2 **Operatori** 22

3.2.1 Assegnazione = ... 22

3.2.2 Operatori aritmetici (+, -, *, /,%) 22

3.2.3 Operatori di assegnazione composti (+=, -=,*=,/=) 22

3.2.4 Operatori di incremento e decremento 22

3.2.5 Operatori di comparazione 23

3.2.6 Operatori di Logici 23

3.3 **Strutture di controllo** 24

3.4 **Cicli: while, do ...while,for** 25

3.5 Esercitazioni - Algoritmi **27**
3.5.1 Integrale di una funzione con il metodo dei rettangoli 27
3.6 ifstream e ofstream: lettura e scrittura su file **30**
3.6.1 Lettura di un file di un numero incognito di quantità 31
3.6.2 Caricare dati in un vettore e ricerca dei max e min assoluti 33
3.6.3 Costruzione di un istogramma 34
3.7 Funzioni **36**
3.7.1 Passaggio di valori a funzioni via value o reference 38
3.7.2 Ricorsività delle funzioni 38
3.7.3 Template di funzione 38

4 Classi 39

5 Dal C++ procedurale al Fortran 41

III Sturmenti di Analisi Dati

6 ROOT 45

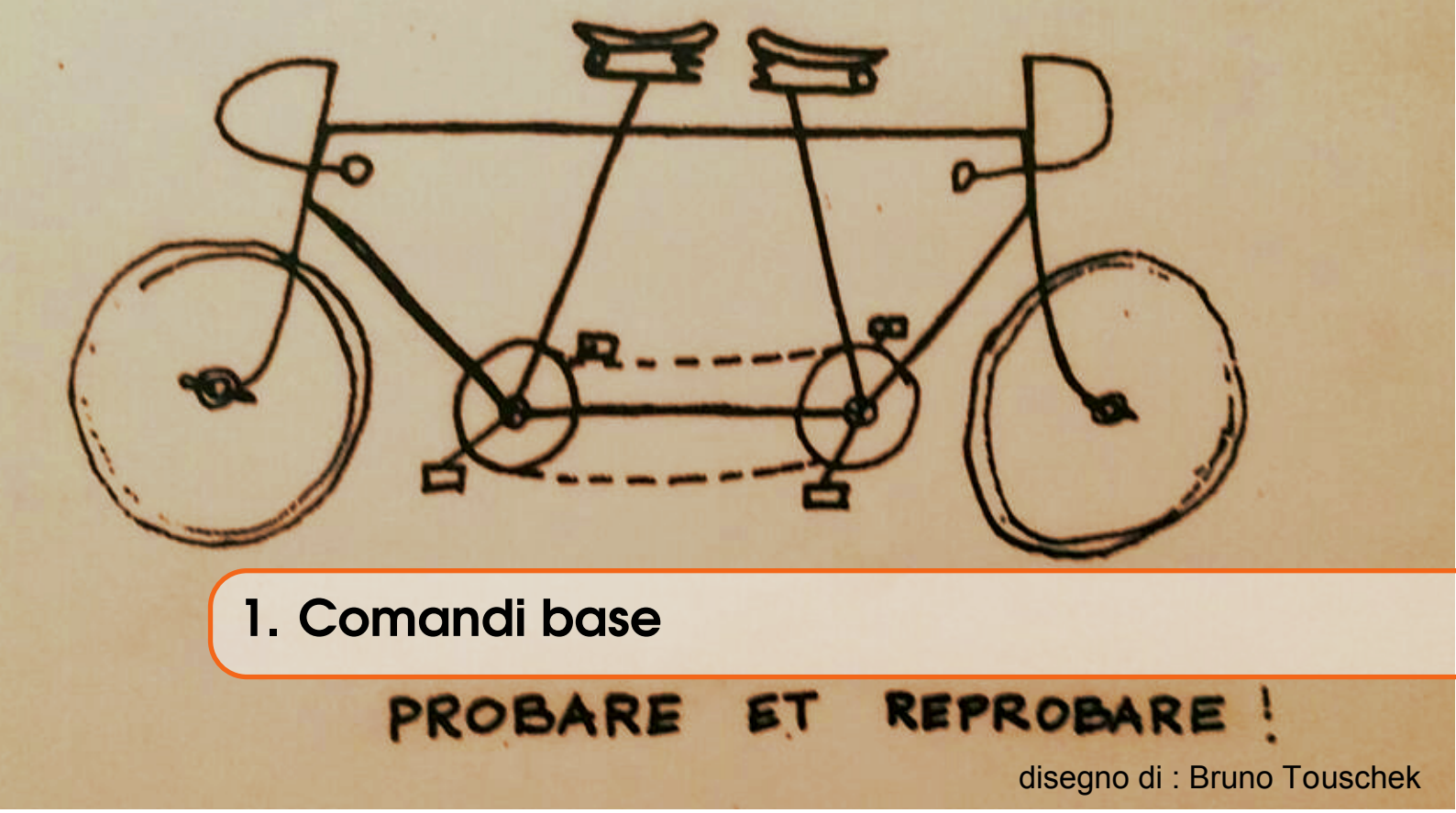
7 Gnuplot 47

Bibliography 49
Books **49**
Articles **49**

Index 51

I Linux

1 Comandi base 9
1.1 Shell-iniziare a muoversi sulla riga di comando
1.2 Compilatori



disegno di : Bruno Touschek

1. Comandi base

Linux è un sistema operativo libero, potente e competitivo con quelli commerciali. Esistono diverse famiglie e distribuzioni, per il corso non si consiglia una versione in particolare, l'unica avvertenza è che la versione installata sia stabile e gradita all'hardware del computer che si utilizza. Nel Laboratorio di Informatica del Corso di Laurea in Fisica la distribuzione in uso è Ubuntu, le macchine sono stabili e ben manutenute.

In questo capitolo verranno descritti i comandi base per iniziare a lavorare sul terminale. Si, assolutamente si, avete letto bene lavoreremo sul terminale e il mouse e l'interfaccia grafica li utilizzeremo pochissimo. In questo momento vi state chiedendo se abbiamo intenzione di trasportarvi nel passato, ma vi assicuro che è tutto l'opposto. L'intento del corso è di portarvi nel futuro. Il passato è fatto di interfacce grafiche accattivanti che vi nascondono ciò che succede realmente nella macchina e che vi condannano all'ignoranza e alla dipendenza. Quindi un po di fiducia e il divertimento non tarderà ad arrivare.

1.1 Shell-iniziare a muoversi sulla riga di comando

In figura 1.1 è presente un dettaglio del terminale dei comandi offerto da un sistema operativo di tipo Linux, in questo caso specifico si tratta di Opensuse. Dal terminale anche detto "shell" è possibile attraverso la chiamata diretta di una serie di comandi (programmi offerti dal sistema

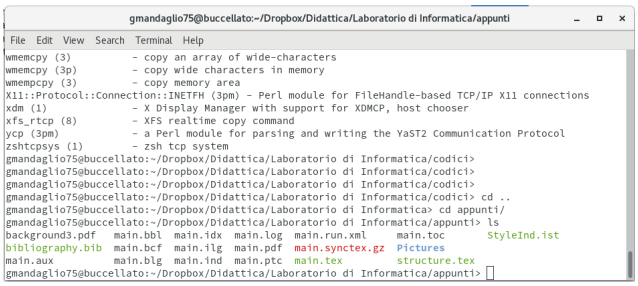


Figura 1.1: Dettaglio del terminale.

operativo - molti ereditati da UNIX) effettuare diverse operazioni: copiare, creare cartelle, muoversi tra le cartelle, leggere documenti, cancellare documenti o cartelle etc.

Ogni comando può avere diverse opzioni, per accedere a queste opzioni bisogna scrivere di seguito il comando spazio-bianco opzione(in genere una lettera) spazio-bianco argomento-su-cui-agisce-il-comando.

Per interrogare il "terminale" circa le opzioni del comando è sufficiente scrivere **man** spazio-bianco nome-del-comando.

Di seguito ne elenchiamo i più comuni e frequentemente usati:

ls stampa a schermo la lista dei file e delle directory contenute nella directory in cui il comando viene invocato. le opzioni -a vi fa vedere anche i file nascosti (su Linux il loro nome inizia con un punto) -l fornisce una lista dei file con più informazioni.

cd acronimo di *change directory* serve appunto a cambiare la directory corrente, a spostarsi dentro la memoria del pc. Per entrare in cartelle e sottocartelle basta separare in fase di comando i nomi delle cartelle con uno slash /. Per scendere di una cartella cd ../.

cp serve per copiare i file. Per copiare le cartelle bisogna utilizzare l'opzione -r (copia ricorsiva). All'interno della stessa cartella bisogna dire al comando il nome del file da copiare e il nome della copia (che può essere differente dal nome del file da copiare) nel caso in cui si copia un file in un altro posto (secondo parametro del comando è il percorso di dove si vuole copiare il documento) il comando automaticamente creerà una copia con lo stesso nome del file di partenza.

mkdir creare una nuova cartella (directory).

rmdir rimuovere una cartella.

rm rimuovere i file. Con l'opzione -r consente di cancellare cartelle e tutto ciò che contengono. Il comando rm -rf è molto potente e pericoloso, consente di cancellare file e cartelle in modo ricorsivo attraverso l'opzione r, mentre non chiede il permesso di cancellare attraverso l'opzione f (force).

pwd vi dirà la posizione della directory corrente.

chmod consente di cambiare i permessi dei file (lettura=r, scrittura=w esecuzione=x). Grazie a questo comando potete rendere eseguibile un documento di testo contente una serie di comandi interpretabili dal terminale che "lanciato" (./nomedelfile invio) eseguirà in sequenza i comandi contenuti nel file (bash script eseguibile). chmod seguito dal meno per rimuovere il permesso, seguito dal + per conferirlo.

mv utile per muove i documenti da una cartella ad un'altra, oppure per cambiare il nome di un file.

tar consente di creare file di archivio utili per conservare i propri dati o per condividerli.

gzip consente di comprimere i file.

gunzip consente di decomprimere i file compressi.

df vi fornisce informazioni sullo stato della memoria del vostro pc.

Questa lista si potrebbe prolungare per diverse pagine, senza però dare mai una descrizione completa di tutte le funzionalità dei vari comandi. Il consiglio migliore per chi comincia a lavorare su un terminale unix-like tipo linux è quello di interrogare il sistema circa le potenzialità del

comando che si intende usare visualizzando a schermo il manuale del comando utilizzando la parola chiave *man* nel seguente modo man-spazio-ilnomedelcomando-invio; e inoltre quello di cercare di imparare più comandi possibili ma non con la smania di un collezionista ma con la voglia di migliorare le proprie possibilità. L'unico modo per memorizzare comandi e opzioni è usarli, usarli e usarli. L'unico modo per apprenderne altri è non usare sempre le stesse cose che si conoscono ma cercare se esistono modi alternativi per fare in modo più efficiente e veloce quello che normalmente facciamo (lo so quando si ha fretta è più comodo usare ciò che si conosce, ma prima o poi ci sarà l'occasione in cui non avremo fretta: bene in quelle circostanze bisogna cercare di essere un po' più curiosi e un po' meno pigri).

1.2 Compilatori

I compilatori il più delle volte risultano già installati su sistemi tipo Linux oppure è possibile scaricarli gratuitamente. Sono disponibili praticamente tutti i compilatori dei linguaggi di programmazione: fortran, pascal, C, C++, perl, phyton, java etc. Il linguaggio di riferimento di questo corso è il C++ e il suo compilatore si invoca attraverso il comando g++ (su mac il comando è cc e la sintassi è la stessa). Verrà inoltre utilizzato il linguaggio fortran e il compilatore di su linux è gfortran. La lettera g che precede questi compilatori sta per gnu. La sintassi e l'utilizzo dei due compilatori è esattamente la stessa.

Il compilatore ha il compito di tradurre il sorgente del programma (un documento di testo contenente l'implementazione del programma) scritto e comprensibile all'uomo in un file comprensibile dalla macchina che chiameremo file eseguibile o eseguibile.

Il compilatore quando viene invocato ad analizzare un codice sorgente esegue sempre un controllo degli errori e nel caso in cui ne dovesse trovare quanlcuno non completerà la compilazione, non produrrà il file eseguibile e vi mostrerà a schermo una serie di messaggi (numero di riga del codice ed errore) che vi aiuteranno ad individuare l'errore/i che avete commesso (vedi fig. 1.2).

```
gmandaglio75@buccellato:~/Dropbox/Didattica/Laboratorio di Informatica/codici> g++ maxmin.cpp
maxmin.cpp: In function 'int main()':
maxmin.cpp:28:1: error: expected ';' before '}' token
 }
 ^
maxmin.cpp:39:31: error: expected ';' before ')' token
 for (int i=0; i<contatore, i++){
                               ^
gmandaglio75@buccellato:~/Dropbox/Didattica/Laboratorio di Informatica/codici>
```

Figura 1.2: Esempio di errori evidenziati dal compilatore. .

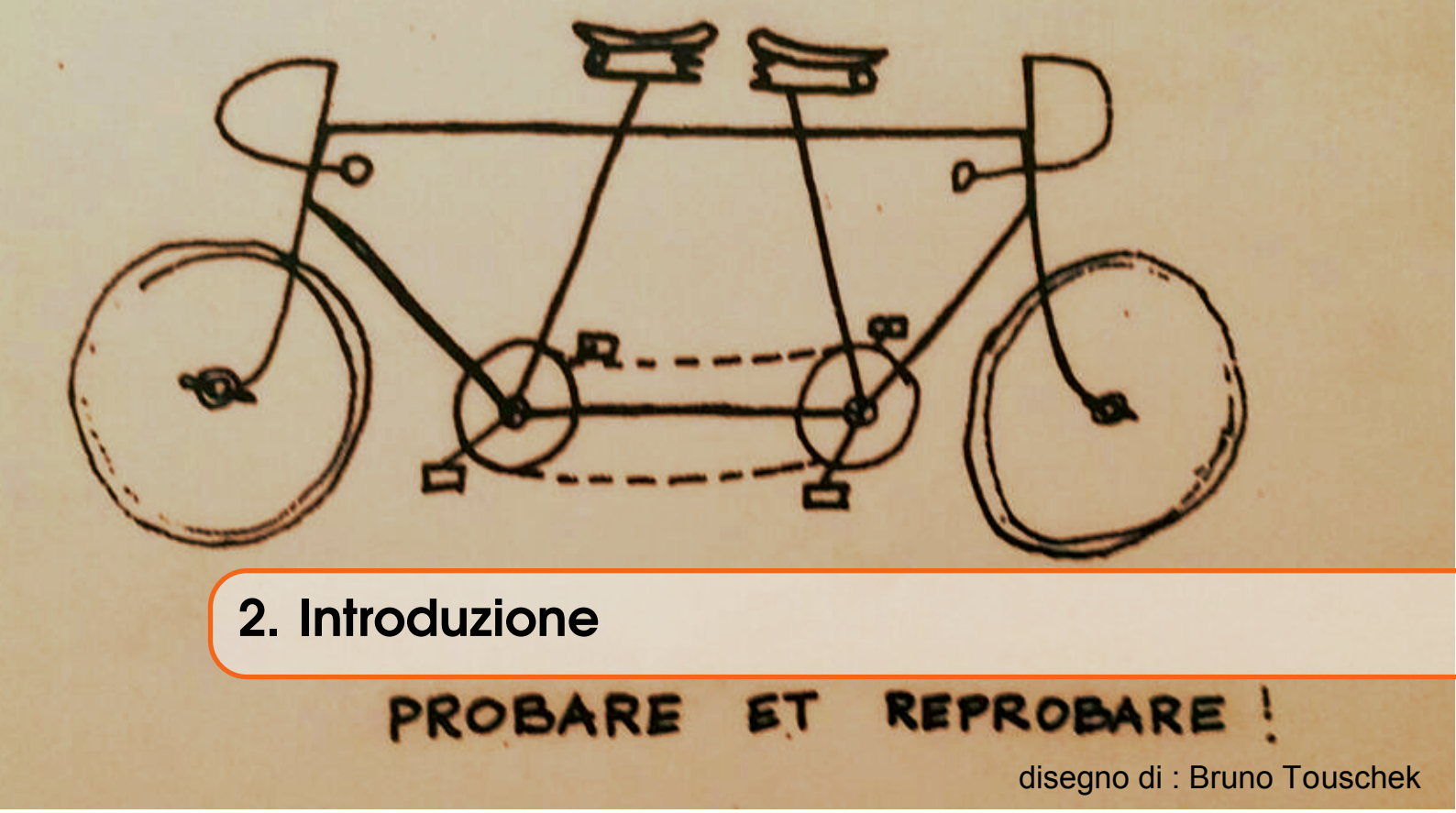
II Programmazione in C++

2 Introduzione 15
2.1 Stuttura di un programma
2.2 Operazioni di input-output I

3 Elementi base per la programmazione 21
3.1 Variabili I
3.2 Operatori
3.3 Strutture di controllo
3.4 Cicli: while, do ...while,for
3.5 Esercitazioni - Algoritmi
3.6 ifstream e ofstream: lettura e scrittura su file
3.7 Funzioni

4 Classi 39

5 Dal C++ procedurale al Fortran 41



disegno di : Bruno Touschek

2. Introduzione

La programmazione è la capacità di far eseguire a una macchina una serie di istruzioni nel modo più efficiente possibile. La sequenza delle istruzioni la chiameremo algoritmo. La correttezza degli algoritmi, l'assenza di errori logici (bug), l'efficienza e la leggibilità del codice rappresentano l'abilità principale del programmatore. Normalmente i programmatori informatici sono formati per implementare codici per soddisfare le esigenze di un committente (un cliente), mentre i programmatori fisici implementano codici per loro stessi o per il gruppo di ricerca con cui collaborano. Sebbene i fisici siano molto apprezzati come programmatori e risolutori di problemi complessi, tuttavia hanno anche la pecca di scrivere codici comprensibili solo a loro e spesso privi di qualsiasi documentazione. Quindi prima di prendere questa brutta abitudine impariamo che i codici devono essere leggibili e ben documentati in modo che anche altri gli possano usare.

Riprendendo il sermone precedente notiamo che ci sono alcune parole che ci suggeriscono che per poter programmare abbiamo bisogno di alcune cose....

Alcune cose:

Algoritmo. Questa parola che sembra complicata, in realtà è una delle cose più comuni della nostra vita. Implementiamo algoritmi in continuazione da sempre: alle elementari alle prese con i primi problemini di algebra pieni di contadini, uova, mercatini etc; quando cuciniamo una frittata; quando guidiamo la macchina; quando scriviamo un messaggio sul cellulare etc. Creiamo algoritmi in continuazione creiamo una sequenza di azioni atte a risolvere un problema o a fare una azione e il pc normalmente siamo noi stessi.

Purtroppo il pc non possiede un cervello come il nostro e non può risolvere i problemi al posto nostro. Si, purtroppo è così!!! I problemi li dobbiamo risolvere noi. Il pc però è veloce, straordinariamente veloce.

Il PC parla una sua lingua e noi ne parliamo un'altra; come possiamo fare a comunicare le nostre idee geniali in modo che il pc possa aiutarci a risolvere il dato problema??

Ci serve un interprete che traduca i nostri codici in un linguaggio comprensibile alla macchina, un linguaggio con delle regole e degli strumenti che ci consentano di implementare i nostri algoritmi.

I compilatori, ovvero gli interpreti, sono dei programmi che hanno il compito di tradurre i nostri codici (codice sorgente) in linguaggio macchina. I compilatori hanno capacità di diagnostica

sulla correttezza sintattica del codice, vi informano degli errori e non procedono alla creazione un programma eseguibile finchè il codice sorgente non viene corretto da tutti gli errori di sintassi. Il compilatore non ci rende immuni dagli errori di logica, quindi scrivere codici corretti è condizione necessaria ma non sufficiente alla risoluzione di un problema con un codice. L'errore più grave che può commettere un programmatore inesperto è subordinare la logica del programma all'esigenza di portare a buon fine un processo di compilazione: espressioni del tipo "ho spostato questo frammento di codice perchè non compilava" possono accorciare la vita della persona che la ascoltano (o almeno possono indurre acidità di stomaco) quindi è moralmente deplorevole.

Nel presente corso il linguaggio eletto allo scopo è il C++, la scelta ha svariate ragioni: è un linguaggio a basso e alto livello, consente di strutturare i programmi in modo che siano orientati agli oggetti, è il linguaggio con il quale sono implementati sistemi operativi, linguaggi ad alto livello come ad esempio python. Si presta bene per imparare a programmare in modo procedurale ed ovviamente orientato agli oggetti.

Alcune lezioni saranno spese per imparare a tradurre i codici procedurali implementati in C++/C in linguaggio Fortran. Il Fortran è un linguaggio molto potente e rivolto al calcolo scientifico. Conoscere la sua sintassi è di grande utilità per poter accedere al patrimonio immenso di codici scritti in questo linguaggio dalla comunità dei fisici e ancora diffusamente utilizzati.

Alla fine del corso, una introduzione alle tecniche fondamentali per l'analisi dei dati verrà affrontata utilizzando il framework root, e il programma gnuplot.

Il corso suggerisce caldamente l'utilizzo di un pc con sistema operativo Linux (qualunque distribuzione va bene), o comunque un sistema operativo tipo Unix. Su linux il compilatore (libero) che verrà utilizzato è g++.

2.1 Stuttura di un programma

Di seguito il listato di un semplicissimo programma in C++ che una volta compilato ed eseguito attraverso la sua versione eseguibile dal pc stampa a monitor la stringa di caratteri "Ciaooooooo!".

```
//con il doppio slash inseriamo commenti nel programma
//i commenti sono ignorati dal compilatore
//i commenti sono utili per rendere più leggibile il codice
// Il primo programma in C++
#include <iostream>// include la libreria iostream

int main()
{
  std::cout << "Ciaooooooo!";
  return 0;
}
```

Per compilare questo codice bisogna che esso sia trascritto in un file con estensione cpp (utile perchè ci dice che abbiamo a che fare con un codice sorgente di tipo C++ e inoltre informa i programmi di editor del tipo di file in questione e l'editor colora in modo molto utile le righe del codice aiutandoci nella programmazione). Poi su un Linux da riga di comando si digita :
g++ nomedelsorgente.cpp
il compilatore g++ per default crea il file a.out che contiene il nostro eseguibile. Per eseguire il programma bisogna digitare sul terminale il comando

```
./a.out
```

e poi fare invio.

Ogni volta che eseguiamo una compilazione in questo modo, g++ sovrascriverà il file a.out con il nuovo eseguibile e quindi il vecchio sarà perso. Se invece si vuole conservare una copia dell'eseguibile può essere utile utilizzare l'opzione di g++ che consente di decidere il nome dell'eseguibile.
g++ -o nome_eseguibile sorgente.cpp (la lettera o sta per output della compilazione)

All'interno del codice sorgente è possibile scrivere dei commenti che sono riservati al programmatore e che non vengono interpretati dal compilatore. I commenti sono preceduti da un doppio slash. I commenti sono molto utili a rendere il codice più leggibile e forniscono una breve documentazione al codice.

Procediamo a commentare il programma riga per riga: prima di ogni cosa troviamo una istruzione che inizia con carattere cancelletto e che segue con il comando include. Questa istruzione serve a dire al preprocessore del compilatore che deve appunto includere nel programma la libreria iostream. Senza l'inclusione della libreria iostream che contiene le funzionalità di input-output non avremmo potuto utilizzare il comando std::cout « "Ciaooooooo!"; e stampare a schermo la stringa "Ciaooooooo!".

Inclusa la libreria, il programma definisce la funzione principale. In C++/C qualunque programma non può prescindere dalla funzione principale, in questo linguaggio tutte le procedure sono funzioni e la funzione che ha il nome main è una funzione speciale e come già detto la più importante. La funzione main ha il compito di coordinare il funzionamento del programma. Usiamo la funzione main per introdurre la struttura di una funzione.

Ogni istruzione deve essere chiusa con **un punto e virgola** ad eccezione dell'inclusione delle librerie o quando raggruppiamo istruzioni utilizzando le parentisi graffe.

```
// int è il tipo di valore che la funzione restituisce
// se la funzione non restituisce nulla si può mettere void al posto di int
// all'interno delle parentesi ci sono le variabili che riceve
//possono essere di diverso tipo come nell'esempio
//oppure non esserci nulla, la funzione non riceve nulla
int nomefunzione (int a, float b, double c, ...etc)
{
//all'interno delle graffe ci sono le istruzioni
istruzione 1;
// tutte le istruzioni chiudono con un punto e virgola
// eccetto l'instruzione #include
istruzione 2;
.....etc;
//alla fine dell'esecuzione, o al suo interno
// l'operazione return restituisce un valore
// del tipo dichiarato ed esce dalla funzione
// se la funzione in questione è la funzione
// principale il programma termina
// se è una funzione qualunque restituisce il dato valore
// alla funzione principale e il programma continua a "girare".
return variabile_che_restituisce;
}
```

Ritorniamo al programma, l'istruzione $std :: cout << "Ciaooooooo!"$; utilizza la classe cout della libreria iostream per stampare a schermo la frase "*Ciaooooooo*!". L'operatore minore-minore $<<$ dice che la stringa "*Ciaooooooo*!", definita attraverso le virgolette (senza virgolette Ciaooooooo

sarebbe una variabile) dal programmatore viene data in input al programma cout e il programma cout lo stampa sullo schermo.

cout è un oggetto che ci consente di scrivere sullo schermo, possiamo scrivere parole o frasi scrivendole tra virgolette, oppure variabili e in questo caso il valore della variabile sarà stampata a schermo. es.

```
std::cout << "il valore del nostro calcolo è = "<< valore<<"\n";
```

Nell'esempio, cout stamperà a monitor se valore fosse uguale a 1.23
il valore del nostro calcolo è = 1.23
e poi sarebbe andato a capo perchè abbiamo messo il comando tra virgolette slash n. Se non chiediamo a cout di andare a capo, cout non lo farà automaticamente e scritture multiple appariranno a schermo sulla stessa riga.

Includendo tra le inclusioni delle varie librerie *#include* e la funzione principale il comando:
using manespace std;
si evita il bisogno di scrivere davanti ai membri della libreria il prefisso *std* ::.

I programmi vengono eseguiti in modo sequenziale dalla prima all'ultima istruzione, quindi leviamoci dalla testa che il seguente pezzo di codice possa restituire il valore 5:

```
int a,b,c;
c=a+b;
a=2;
b=3;
cout<<"non restituisce 5 manco se ti metti a piangere :P  ="<< c <<endl;
```

il codice corretto sarebbe:

```
int a,b,c;
a=2;
b=3;
c=a+b;
cout<<"grazie così va bene, la somma di a e b è  ="<< c <<endl;
```

2.2 Operazioni di input-output I

Il programma seguente lo utilizziamo per introdurre l'uso delle variabili. Nell'esempio le variabili si chiamano a,b, e results e sono tutte variabili di tipo intero. Il programma assegna i valori alle due variabili (5 e 2), poi somma la quantità 1 alla variabile a e poi fa la differenza tra a e b e assegna questo valore alla variabile result. In fine stampa a monitor il risultato della differenza attraverso la variabile result.

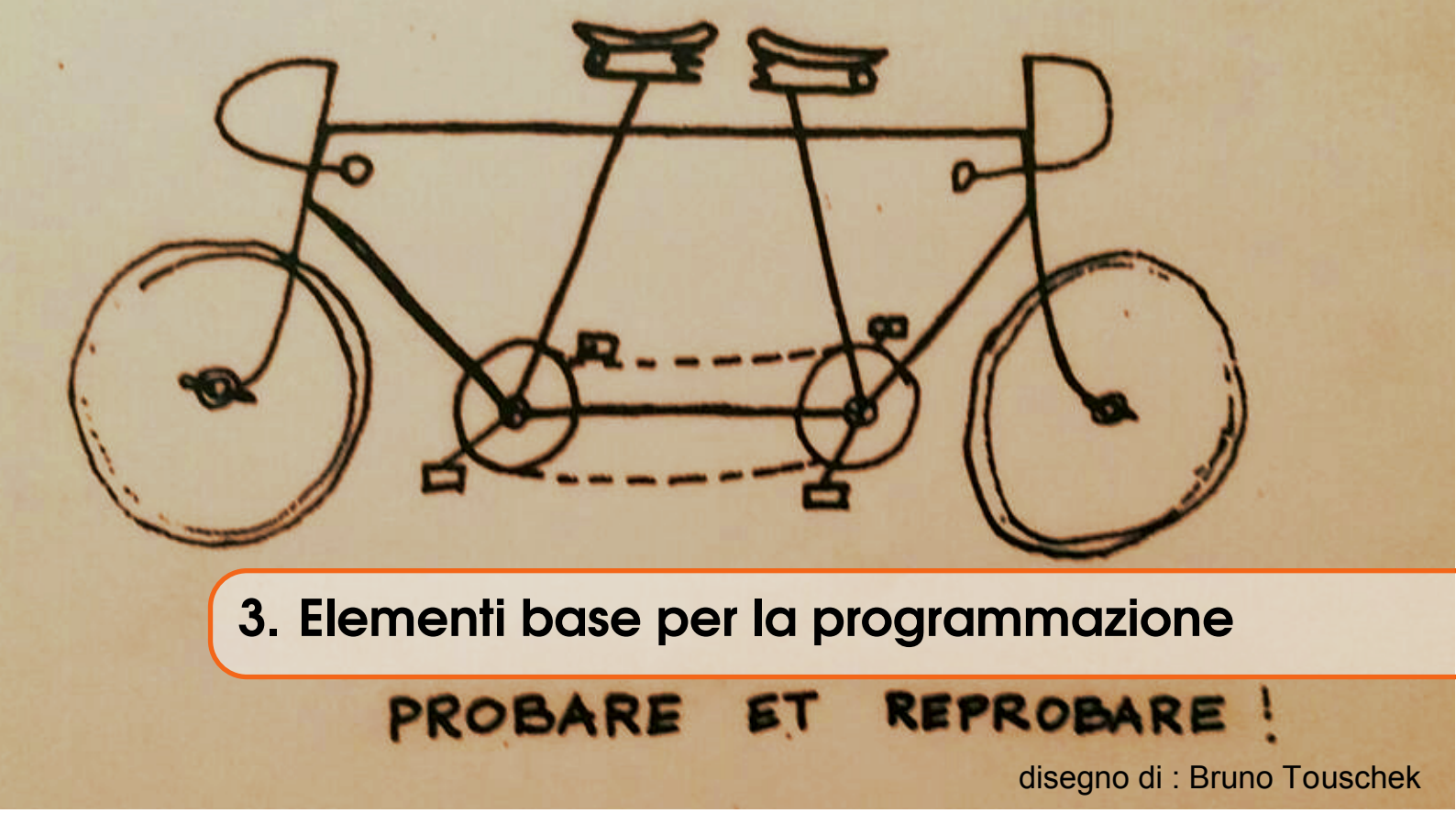
```
// operating with variables

#include <iostream>
using namespace std;

int main ()
{
  // declaring variables:
  int a, b;
  int result;
```

```
  // process:
  a = 5;
  b = 2;
  a = a + 1;
  result = a - b;

  // print out the result:
  cout << result;

  // terminate the program:
  return 0;
}
```

Ora immaginiamo di voler confezionare un programma capace di chiedere ad un utente di fornirgli da riga di comando due numeri, e lui in cambio li somma e restituisce a schermo il risultato. Per fare questo ci serve un qualchecosa che operi al contrario di cout. Il metodo che fa al nostro scopo è cin. cin funziona sintatticamente in modo equivalente a cout solo che i minore-minore si trasformano in maggiore-maggiore ad indicare che cin prende qualche cosa dal terminale e lo assegna ad una varibile nel programma. In pratica cin e cout si comportano come due messaggeri che fanno parlare il mondo del programma con quello esterno del terminale. Più aventi presenteremo anche i loro fratelli omologhi del C printf e scanf, utilizzabili nel C++ che assorbe completamente C.
es. cin»variabile; //assegna ad variabile il valore digitato al terminale dopo l'invio
es. cin»variabile1»variabile2; // in questo caso due valori. Per passarli a cin, abbiamo due strade, digitiamo un valore poi invio e così per il secondo oppure digitiamo entrambi i valori separati da uno spazio bianco e poi invio.

```
// operating with variables

#include <iostream>
using namespace std;

int main ()
{
  // declaring variables:
  int a, b;
  int result;

  // process:
  cout << "potresti darmi un numero?"<<endl;
  //endl è un modo alternativo per andare a capo
  cin  >> a;
  cout << "potresti darmi un'altro numero?"<<endl;
  cin >> b;
  cout << "Grazie!"<<endl;

  result = a n b;

  // print out the result:
```

```
  cout << "ecco il risultato della somma =" <<result<<endl;
  cout << "ecco il prodotto occhio al codice ="<< a*b<<endl;

  // terminate the program:
  return 0;
}
```



disegno di : Bruno Touschek

3. Elementi base per la programmazione

3.1 Variabili I

Le variabili in qualunque linguaggio di programmazione possono essere di diverso tipo: intere, reali, complesse, caratteri, stringhe (insiemi di caratteri) etc. Le variabili vanno sempre dichiarate, questa operazione è fondamentale perchè bisogna prenotare le locazioni di memoria capaci di ospitare il dato. Se le variabili non vengono dichiarate il compilatore segnalerà la cosa come un errore: il C++/C non sono dotati di un sistema automatico di assegnazione delle variabili (phyton ad esempio è dotato di un sistema di assegnamento automatico, ma phython è un linguaggio al alto livello e tutto quello che ci risparmia è dovuto a implementazioni che fanno il lavoro al posto nostro. python è implementato in c++).

Il codice seguente:

```
#include <iostream>// include la libreria iostream

int main()
{
  pi_greco=3.14;
  return 0;
}
```

produrrebbe il seguente errore in fase di compilazione

```
gmandaglio75@buccellato:~> g++ prova.cpp
prova.cpp: In function ‘int main()’:
prova.cpp:6:3: error: ‘pi_greco’ was not declared in this scope
   pi_greco=3.14;
   ^~~~~~~
```

Come si può vedere in tabella 3.1 ci sono diversi tipi di variabile che possono essere utilizzate dentro i nostri codici. É importante notare che la grandezza delle variabili è considerevole nelle

Gruppi	Tipi di carattere	note
Caratteri	char	1 byte, almeno 8 bits
	char16_t	non più piccolo di un Char, almeno 16 bits.
	char32_t	non più piccolo di un char16_t, almeno 32 bits.
	wchar_t	il più grande tipo di carattere supportato.
Tipo intero con segno	signed int	da -2147483648 a 2147483648 (32 bit)
	signed long int	da -2147483648 a 2147483648
	signed long long int	da -9223372036854775808 a 9223372036854775808
Tipo reale	float	da 1.17549×10^{-38} a $3.40282\times10^{+38}$
	double	da 2.22507×10^{-308}a $1.79769\times10^{+308}$
	long double	da 3.3621×10^{-4932} a $1.18973\times10^{+4932}$
Tipo Booleana	bool	assume i valori 1 - 0 o “true” - “false”
Tipo vuoto	void	nessuno spazio di memoria occupato

Tabella 3.1: Lista di variabili disponibili in C++, nome e dimensioni.

variabili di tipo *long*, ma non infinita. Il concetto di infinito è per gli esseri umani non per le macchine.

3.2 Operatori

3.2.1 Assegnazione =

L’operatore di assegnazione è =. Lo abbiamo già utilizzato negli esempi precedenti. É importante sottolineare che questo operatore esegue solo l’operazione di assegnazione ovunque esso venga invocato, non comparazione, non risponde a nessuna domanda! Scivere che a=b; o a =25; significa che la variabile a assumerà lo stesso valore della variabile b oppure il valore 25 nel secondo caso.

3.2.2 Operatori aritmetici (+, -, *, /,%)

Gli operatori somma, sottrazione, prodotto(simbolo asterisco *), divisione (simbolo slash /), e quoziente (simbolo %) ci consentono di lavorare su numeri e variabili. Bisogna ricordarsi che gli operatori abbediscono ad un ordine gerarchico che dá a prodotto/divisione/quoziente priorità di esecuzione sulle operazioni di somma e sottrazione. Quindi la variabile pippo nel seguente frammento di codice assumerà il valore 11 piuttosto che 21.

```
int pippo = 5 + 2 * 3;
```

L’operatore quoziente % restituisce il resto della divisione tra due numeri. Questa operazione è molto utile se vogliamo stabilire se un numero intero memorizzato in una variabile sia pari o dispari, per fare questo basta verificare se il valore restituito dall’operazione variabili%2 sia pari a zero (pari) o a 1 (dispari).

3.2.3 Operatori di assegnazione composti (+=, -=,*=,/=)

In C++ è possibile comporre gli operatori di assegnazione con altri creando delle scritture abbreviate come nella tabella seguente:

3.2.4 Operatori di incremento e decremento

Gli operatori ++ e −− sono detti operatori di incremento e di decremento, nel senso che aumentano o diminuiscono di una unità la variabile alla quale vengono applicate.

```
x++; // equivale a x = x +1;
x--; // equivale a x = x -1;
```

espressione	equivalente a...
y += x;	y = y + x;
x -= 5;	x = x - 5;
x /= y;	x = x / y;
x *= y;	x = x * y;

Tabella 3.2: Equivalenza tra assegnazioni composte.

é importante notare il verso dell'applicazione dell'operatore nel caso in esso venga utilizzato nelle espressioni:

```
x=20; //
y = x++; // in questo caso y è uguale 20 e x a 21
//mentre!!!!!!!!!!
y = ++x;  //in questo caso y e x sono uguali a 21
```

Nel primo caso prima viene effettuata l'operazione di uguaglianza e poi quella di incremento producendo una differenza tra le due quantità alla fine dell'operazione, nel secondo caso invece prima si incrementa e poi si eguaglia.

3.2.5 Operatori di comparazione

Gli operatori di comparazione ci aiutano a stabilire la vericidità di una comparazione tra variabili e grandezze. Se la relazione scritta è soddisfatta restituirà il valore *vero* altrimenti il valore *falso*. Queste espressioni saranno di fondamentale inportanza per le strutture di controllo e per il ciclo *while* che vederemo in seguito.

operatore	descrizzione
==	Uguale
!=	Diverso
<	Minore di
>	Maggiore di
<=	Minore uguale a
>=	Maggiore uguale a

Tabella 3.3: Operatori di comparazione.

3.2.6 Operatori di Logici

Gli operatori logici ci consentono di combinare delle condizioni, e a seconda della loro vericiticità restituiranno un valore vero o falso. L'operatore OR è realizzato legando due condizioni con un *doppio pipe* mentre la codizione AND con una *doppia e commerciale*, mentre la negazione è ottenuta con il punto esclamativo.

operatore	descrizzione
&&	AND
||	OR
!	NOR

Tabella 3.4: Operatori di logici principali.

L'operatore AND restituisce un valore vero se e soltanto se entrambe le condizioni sono vere, mentre l'OR restituisce vero se almeno una delle due condizioni è vera.

3.3 Strutture di controllo

Le strutture di controllo ci consentono di inserire nei nostri programmi la possibilità di scegliere se eseguire alcune istruzioni oppure altre a seconda della veriticità di una condizione. La condizione può essere una variabile logica booleana, o una espressione composta con gli operatori di comparazione, oppure il valore (0 o 1) restituito da qualcuno (va bene, va bene, non vi seccate: questo qualcuno potrebbe essere una classe o una funzione ma lo vedremo in seguito). Le parole chiave che ci consentono di attivare una tale struttura sono if ed *else*, la prima significa *se* mentre la seconda *altrimenti*. L'opzione *else* non è obligatoria, e se non si usa l'istruzione if porrà condizione alla istruzione o all'insieme di istruzioni racchiuse tra graffe che la seguono. Se dopo l'if o dopo l'*else* se si vuole far seguire una sola istruzione allora in questo caso le parentesi graffe possono essere omesse. La condizione controllata dall'if deve essere racchiusa tra parentesi tonde, mentre l'*else* eseguendo ciò che la segue se la condizione dell'if non è "vera" non ha bisogno di una sua condizione tra parentesi tonde.

```
\\struttura del costrutto if ... else

if(condizione) {
istruzione 1;
istruzione 2;
....

istruzione n;
}
else{
Istruzione 1;
Istruzione 2;
....

Istruzione n;
}
```

Un primo esempio semplice, che richiama l'uso dell'operatore quoziente % è quello di determinare se un numero dentro un codice sia pari o dispari.

esempio:

```
int pippo;
.....
......
....... (a un certo punto pippo assumerà un valore)

if(pippo%2==0)
cout<<"pippo è pari";
else
cout<<"pippo è dispari";
```

Un secondo esempio, familiare a masticatori di numeri come i fisici è quello in cui si vuole indicare l'interno di un intervallo oppure il suo esterno. Un caso molto semplice che possiamo considerare è quello di un intervallo fissato da due valori su un asse di un sistema di riferimento cartesiano. Grazie a questo esempio possiamo esercitarci con la struttura di controllo e nel definire la condizione della struttura con gli operatori di confronto e gli operatori logici. L'esempio può

essere formulato in modo equivalente utilizzando l'operatore AND (che individua l'interno) oppure l'OR (che individua l'esterno).
esempio AND:

```
// immaginiamo su un ascissa un intervallo tra i valori 2 e 7
float x;
....
.....
....
if((x>=2) && (x<=7))
cout<< "x è interno all'intervallo"<<endl;
else
cout<< "x è esterno all'intervallo"<<endl;
```

esempio OR:

```
// immaginiamo su un ascissa un intervallo tra i valori 2 e 7
float x;
....
.....
....
if((x<2) || (x>7))
cout<< "x è esterno all'intervallo"<<endl;
else
cout<< "x è interno all'intervallo"<<endl;
```

3.4 Cicli: while, do ...while,for

Nella programmazione una operazione molto utile e ricorrente è quella di ripetere una istruzione o un gruppo di istruzioni un certo numero di volte.

Un esempio molto semplice che ci può aiutare a comprendere l'utilità dei cicli è l'operazione di somma. É vero che se dobbiamo sommare due numeri non è un grande problema, e neanche se fossero 10 se invece fossero 20 sarebbe orribilmente noioso ma cela potremmo fare ma se le quantità che dobbiamo sommare fossero 100, o 1000 o 1000000 etc. In questo caso il modo più semplice per non scrivere una istruzione lunga un chilometro, e cavarsela con poche righe di codice è ricorrere a un ciclo. L'istruzione potrebbe consistere di una operazione che somma un addendo alla volta e lo accoumuli in una variabile contenitore inizializzata a zero (elemento neutro dell'operazione di somma) prima dell'inizio del ciclo.

Come esempio implementeremo un programma che ci chiede il voto che abbiamo conseguito in tutti i corsi che abbiamo superato e una volta che abbiamo finito di darglieli ci restituirà la media:

```
// operating with variables

#include <iostream>
using namespace std;

int main ()
{
  // declaring variables:
  float somma, voto;
  int n_materie_superate = 0;
```

```
   somma = 0;

  cout << "questo programma calcola la media"<<endl;
  cout << "dei voti che hai conseguito"<<endl;
  voto=1; //per innescare il ciclo while
  while (voto > 0){
  cout << "dimmi il voto o scrivi 0 per chiudere"<<endl;
  cin  >> voto;
  if( voto > 0)
  {
   somma = somma + voto ; // avremmo potuto scrivere somma + = voto;
   n_materie_superate++; // incrementiamo di una unità
  }
  }
  // print out the result:
  cout << "La tua media è =" <<somma /n_materie_superate <<endl;
  return 0;
}
```

Nell'esempio precedente abbiamo implementato un ciclo utilizzando il costrutto del *while*, questa parola significa $finch$ e consente di ripetere le istruzioni che lo seguono finchè la condizione contenuta tra le parentesi tonde che lo seguono è soddisfatta (o in altre parole è vera). Il ciclo *while* consente di costruire cicli che non si conosce a-priori la durata, e come abbiamo visto nell'esempio precedente abbiamo avuto bisogno di un "innesco" che consentisse al ciclo di partire. Una volta partito, le sorti del ciclo dipendono dalle istruzioni che lo seguono. Questo costrutto è molto potente, e come dice lo zio dell'Uomo Ragno "da grandi poteri discendono grandi responsabilità" e quindi è anche molto pericoloso. A parte gli scherzi, se il *while* non è implementato in modo corretto può trasformarsi in un ciclo infinito dal quale non è possibile uscire.

Se volessimo ovviare al problema dell'innesco potremmo usare il costrutto $do-while$ il quale in ogni caso almeno una volta eseguira le istruzioni che il costrutto contiene:

```
do{
istruzione 1;
istruzione 2;
....
....
....
}while(condizione)
```

Se invece vogliamo impletare un ciclo e conosciamo a-priori il numero di volte che lo vogliamo ripetere, allora in questo caso possiamo utilizzare il costrutto del for. Il costrutto del ciclo è il seguente:

```
for(int indice; indice< fine; indice++){
istruzione 1;
istruzione 2;
....

....
....
}
```

Anche per il *for* se l'istruzione che si deve ripertere è una sola si possono omettere le parentesi graffe.

Le implementazioni dei cicli sono equivalenti tra di loro. Nell'esempio che segue il codice stampa a schermo i numeri da 0 a 9 utilizzando in modo equivalente i due costrutti.

```
for(int i =0 ; i<10; i++) cout<<i<<endl;
//oppure
int i =0;
while(i<10){cout<<i<<endl; i++;}
```

Un esempio di utilizzo del ciclo *for* utile quando si vuole generare una sequenza di numeri è dato dal seguente codice che stampa a schermo 20 valori a passo costante da 0 a 3.14.

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

main (){
float pi=3.14;
float n=0;
cout<<"Sequenza dei numeri da 0 a pi-greco"<<endl;
for(int i=0; i<20; i++){
n=n+(pi/20);
cout<<n<<endl;
}
return 0;
}
```

3.5 Esercitazioni - Algoritmi

3.5.1 Integrale di una funzione con il metodo dei rettangoli

In questo paragrafo utilizzeremo un esercizio svolto in classe per fare un po di riassunto delle cose finora apprese. L'esercizio consiste nel calcolare l'integrale della funzione $f(x) = x^2$ nell'intervallo tra 0 e 5 con il metodo numerico dei rettangoli. Questo metodo numerico è probabilmente il più semplice per stimare numericamente un integrale definito. Esso consiste nell'approssimare l'area sottesa alla curva con la somma dei rettangoli costruiti in modo che la loro base sia pari all'intervallo di integrazione diviso per il numero di divisioni che vogliamo utilizzare per il calcolo e la loro altezza sia pari al valore assunto dalla funzione nel valore medio del sotto intervallo in considerazione. In Figura 3.1 sono rappresentati la curva in questione e i rettangoli costruiti per 10, 20, 30, 40 divisioni dell'intervallo di integrazione. Dalla figura intuiamo che all'aumentare del numero di divisioni aumenta la precisione dell'integrale. Come funziona questo metodo? L'idea di base consiste nel fatto che l'area del rettangolo rappresenta una buona stima di quella ottenuta sostitendo alla base superiore del rettangolo la curva descritta dalla funzione. Se poniamo attenzione all'intersezione tra curva e rettangolo notiamo la formazione di due figure simili a "triangoli", uno superiore alla curva che introduce una sovrastima all'area e uno inferiore che la sottostima. L'algoritmo sfrutta il fatto che queste due aree si conpensano.

Sfruttiamo questo esempio per introdurre l'utilizzo di una nuova libreria. Quando abbiamo bisogno di utilizzare funzioni matematiche come seno, coseno, tangente etc o operazioni tipo l'elevamento a potenza abbiamo bisogno di caricare la libreria che ci abilita a poter utilizzare queste

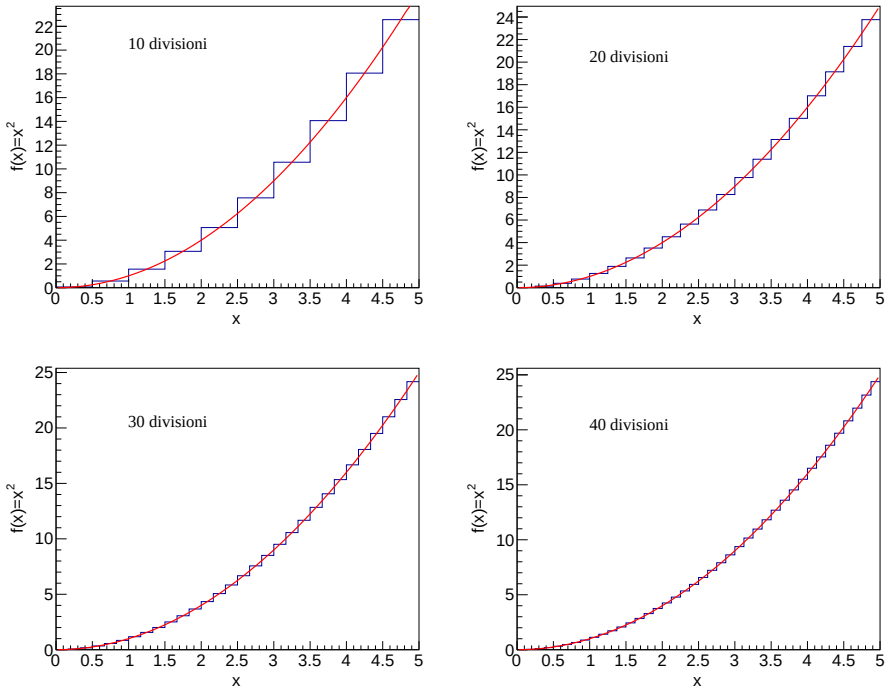


Figura 3.1: Rappresentazione del metodo dei rettangoli per l'integrale della funzione $f(x) = x^2$ tra 0 e 5, utilizzando 10, 20, 30 e 40 divisioni.

funzionalità del linguaggio di programmazione. La libreria matematica standard si chiama *cmath*, e nel nostro esempio ci avvaleremo della funzione *pow* (diminuitivo di power) per elevare alla seconda potenza una variabile. Il costrutto di *pow* è il seguente:

```
y = pow(base, potenza);
```

Notare che la potenza può essere anche un numero razionale, ciò ci consente di utilizzare questa funzione per definire le operazioni di radice a qualunque ordine: potenza =1/2 - radice quadrata; potenza = 1/3 radice cubica; potenza = 7/5 - radice quinta di potenza 7 etc.

NOTA: Le funzioni trigonometriche sin(var); cos(var); tan(var) etc vogliono come informazione l'angolo espresso in radianti, quindi se avete dati espressi in angolo bisogna prima convertirli in radianti dividendo l'angolo per 180° e moltiplicando per π-greco.

Di seguito i codice implementati durante il corso da due allieve, uno con una implementazione standard e uno con una scrittura sintetica del ciclo *for*:

```
//integrale di x^2
#include <iostream>
using namespace std;
main (){
float a=0, b=5;
float x; //misura dell'intervallo
float m; //punto medio di ogni intervallo
float Area=0;
x=(b-a)/100;
m=a+x/2;
for(int i=0; i<100; i++){
```

```
Area=Area+(x*(m*m));
m=m+x;
}
cout<<Area<<endl;
return 0;}
```

Nel codice seguente è molto bella la implementazione del ciclo *for* proposta da una studentessa del corso, probabilmente un esperto di programmazione potrebbe obbiettare che un codice compatto e furbo potrebbe essere poco leggibile, ma nonostante l'obbiezione leggittima ho voluto mostrare come esempio questa implementazione perchè anche la creatività ha un suo valore e deve essere incoraggiata, anche se poi un collega che non riesce a capire cosa abbiamo scritto ci dedicherà una litania di benedizioni.

```
#include <iostream>
#include <cmath>
using namespace std;
float somma=0;
float x=0;
float a=0;
float b=5;
float m;
int main() {
m=(b-a)/7527;
x=m/2;
for(x;x<=5;x=x+m)
somma= somma+m*pow(x,2);
cout<<"L'integrale è  "<<somma<<endl;
return 0;
}
```

Ricerca dinamica del numero di divisioni

L'esercizio del calcolo dell'integrale di una funzione semplice e regolare come una parabola ci fornisce la possibilità di implementare un nuovo codice, molto divertente, che ha bisogno dell'implementazione di un ciclo *while* per rispettare le specifiche del problema.

Il nuovo esercizio consiste nello scrivere un programma capace di stimare il numero minimo di divisioni dell'intervallo di integrazione per raggiungere un certo grado di precisione. La funzione in oggetto è molto semplice, ma anche fosse stata più complicata a patto di essere capaci di integrarla analiticamente, possiamo facilmente integrarla in modo esatto e a questo punto spostare l'oggetto del nostro desiderio dall'integrazione alla precisione dell'integrazione.

```
#include <iostream>
#include <cmath>
using namespace std;
int main (){
float a=0, b=5;
float j=1;
float x;
float m;
float Area=0;
while((fabs(125./3.-Area))>0.001){
```

```
j++;
x=(b-a)/j;
m=a+(x/2.);
Area=0;
for(int i=0; i<j; i++){
Area=Area+(x*m*m);
m=m+x;
}
}
cout<<"L'area e': "<<Area<<endl;
cout<<"Il numero di intervalli minimo e': "<<j<<endl;
return 0;
}
```

3.6 ifstream e ofstream: lettura e scrittura su file

Per leggere e scrivere su file il c++ fornisce due strumenti appartenenti alla librerie fstream[1] che si chiamano ifstream e ofstream. É facile memorizzare che ifstream serve le leggere, infatti la i sta per input e la f per file, mentre ofstream serve per scrivere su file, la o ovviamente sta per output.

Mostriamo di seguito il costrutto dei due operatori, prima iniziamo dall'operatore di lettura:

```
ifstream leggimi;
leggimi.open("nomedelfiledilettura.dat");
float dato;
leggimi>>dato;
cout<<"ecco cosa abbiamo letto: "<<dato<<endl;
leggimi.close();
```

Poi mostriamo quello dell'operatore di scrittura:

```
ofstream scrivimi;
scrivimi.open("nomedelfilediscrittura.dat");
for(int i=0;i<100;i++)
scrivimi<< i <<endl;
scrivimi.close();
```

Osservando i due costrutti possiamo notare che i due operatori differiscono per il nome e la loro funzione, ma il loro utilizzo è molto molto simile. Infatti la sintassi di apertura e chiusura del file[2] è la stessa mentre l'operatività degli operatori è identica a quella delle operazioni di input-output da/su terminale cin e cout attraverso gli operatori $>>$ e $<<$.

Ogni operazione di lettura o di scrittura muove il punto di lettura al succevo valore che deve essere letto o scritto: in altre parole il computer legge o scrive in maniera progressiva nel file. Per "riavvolgere il nastro" ovvero per spostare il punto di lettura/scrittura ci sono apposite funzionalità (seekg, seekp).

Le classi ifstream e ofstream hanno diverse funzionalità alle quali si accede attraverso l'operatore . (si proprio così l'operatore punto). Abbiamo visto in azione già l'operazione di apertura e

[1] NOTA BENE: ci dobbiamo ricordare di mettere in testa la scrittura #include <fstream> altrimenti non possiamo usare questi strumenti.

[2] Nota bene: Chiudere il file è importantissimo. Lasciare un file aperto significa dare accesso al sistema operativo alle locazioni di memoria riservate al documento con la possibilità che questo possa essere danneggiato.

chiusura del file, e negli esempi che seguono vedremo le operazioni eof() che ci dice se si è arrivati alla fine del file o meno, le funzioni clear(), seekg e seekp che servono per spostare il punto di lettura o scrittura del file (li useremo per riportare il punto di lettura all'inizio del file dopo aver effettuato una prima lettura).

3.6.1 Lettura di un file di un numero incognito di quantità

Una operazione molto utile è quella di essere capaci di leggere un file di dati il cui numero ci è incognito. A questo punto potremmo fare l'obiezione che è difficile immaginare una simile evenienza. In realtà non è così. Immaginiamo di avere il compito di analizzare dati che provengono da una stazione di monitoraggio di quantità fisiche, per esempio una stazione che monitora la temperatura, o la pressione o la radiattività o etc e che salva queste letture su un file di testo e che effettua una lettura ogni certa quantità di tempo. In un certo momento della giornata scarichiamo il file contenete questi dati e iniziamo ad analizzarli.

Il numero di dati contenuti non ci è noto, e noi non abbiamo il tempo e il modo di aprire il documento e di contare il numero di dati che il file contiene. La soluzione è scrivere un programma capace di farlo in maniera autonomatica, di fornirci il numero di dati che abbiamo letto, e quindi procedere con delle semplici analisi.

lettura del X-file

Per risolvere questo problema faremo uso di un ciclo *while* che ci consente di implementare un ciclo con numero di ripetizioni variabili e dipendenti dalla condizione che lo governa e la funzione eof()[3] dell'operazione ifstream che ci dice se ci troviamo alla fine del file oppure no. L'operazione eof() risponde in modo diretto alla domanda, nel senso che restituisce 1 (vero) se è arrivato alla fine e 0 (zero) in caso contrario. Per usare questo comportamento come condizione per governare il ciclo *while* dobbiamo usare l'operatore punto esclamativo (!) per negare la risposta della funzione eof() in maniera tale che quando lui ci dice che non è alla fine noi diciamo al *while* di effettuare un'altra lettura e quando invece ci dice che è arrivato alla fine noi diciamo al *while* di uscire dal ciclo.

Riportiamo di seguito il frammento di codice per la lettura del file e il conteggio del numero di letture:

```
ifstream leggimi;
leggimi.open("Xfile.dat");
double temperatura;
int NumerodiConteggi=0;
while (!leggimi.eof())
leggimi>>temperatura;
if (!leggimi.eof()) {
cout<< temperatura << endl;
NumerodiConteggi++;
}
cout<< il numero di conteggi è <<NumerodiConteggi<<endl;
```

Commentiamo brevemente il codice che abbiamo scritto: prima di tutto abbiamo creato l'operatore di lettura, aperto il file e creato le variabili su cui copiare la singola lettura e su cui effetturare un conteggio (inizializzata a 0); dopo di che il ciclo *while* verifica se la sua condizione che la governa è vera oppure no. Se è vera c'è almeno un dato dentro il file. A questo punto entriamo dentro il ciclo e procediamo a leggerlo procedura per procedura: leggiamo un valore e

[3] eof sta per end of file.

lo copiamo nella variabile temporanea "temperatura", controlliamo sull'if se è vero che abbiamo letto un valore, lo stampiamo a monitor e poi incrementiamo il contatore, ripassiamo la palla al *while* e così via. Quando processo legge l'ultimo dato valido del file continuerà a soddisfare la condizione del *while* entrerà nel ciclo ed effettuerà una lettura che fallirà perchè non c'è un altro dato da leggere, la condizione if non consentirà di stampare nulla a schermo ne di incrementare il contatore, dopo di che si esce definitivamente dal *while* e si stampa a monitor il valore esatto del numero dei lettura.

Il processo di lettura non interpreta gli spazi vuoti o gli "a capo" e non fa differenza tra file in cui i dati sono scritti in modo ordinato o disordinato, il processo legge elemento ad elemento. Se un file è ordinato e lo vogliamo leggere noi sfruttando questa informazione è nostra responsabilità adottare una strategia capace di utilizzarla al meglio.

Esempio se i dati sono ordinati su due colonne dove sulla prima colonna mettiamo una quantità fisica indipendente e sulla seconda una quantità dipendente dalla prima basta scrivere un codice in questo modo:

```
ifstream leggimi;
leggimi.open("Xfile.dat");
double temperatura, Energia;
while (!leggimi.eof())
leggimi>>temperatura>>Energia;
if (!leggimi.eof()) {
cout<< temperatura << "  "<< Energia<< endl;
}
```

Visto che siamo capaci di accedere ai dati e di stamparli a monitor, possiamo avventurarci nello studio degli stessi e provare a calcolare la media e lo scarto quadratico medio utilizzando le seguenti formule

$$< T >= \sum_{i=0}^{N} \frac{t_i}{N}$$

$$\sigma = \sqrt{\sum_{i=0}^{N} \frac{(< T > -t_i)^2}{N}}$$

e con il seguente codice

```
//calcola la somma e la media

ifstream leggimi(nome);   //legge dal file dati.dat
double a;   //variabile di copia provvisoria
double somma =0;  //accumulatore
int NumerodiLetture = 0; //contatore
while(!leggimi.eof()){
leggimi>>a;   //attribuisce i valori del file alla variabile a,
              //finchè non siamo arrivati alla fine del file
somma=somma+a;
if(!leggimi.eof()){
cout<<a<<endl;      //stampa i valori del file
NumerodiLetture++;
}
}
```

```
media=somma/NumerodiLetture;   //calcola la media
cout<<endl;
cout<<"Hai stampato "<<NumerodiLetture<<" numeri."<<endl;
cout<<"La somma dei numeri stampati e' "<<somma<<".\n";
cout<<"La media dei numeri stampati e' "<<media<<" \n";
leggimi.close();  //chiusura del file
```

3.6.2 Caricare dati in un vettore e ricerca dei max e min assoluti

Nella sessione precedente abbiamo imparato a leggere un file con un numero di dati incogniti e a farci dire il numero di letture. Ora scriveremo un codice capace di caricare i dati in un vettore di dimensione pari al numero di letture e di dirci qual'è il massimo assoluto e il minimo assoluto tra i valori letti.

Per creare un vettore delle dimensioni giuste, capace di contenere i nostri dati, prima di tutto dobbiamo conoscere quanto è questa dimensione e per avere questa informazione ci sono diverse opzioni: la conosciamo (evviva, fine); apriamo il file e li contiamo (che noia!!!!!); le contiamo con il nostro codice (che forti che siamo!!!). La conoscenza di questo numero è importante perchè se imponiamo una dimensione inferiore a quella necessaria il programma di lettura cercherà di scrivere i dati in allocazioni di memoria non riservate al vettore e otterremo un errore nella esecuzione del programma (NOTA BENE - non in fase di compilazione ma in fase di esecuzione) se invece mettiamo prudentemente un numero enorme probabilmente potremmo essere al sicuro da questo problema (ma non è detto) ma violeremmo il principio di richiesta minima di risorse (in questo caso memoria) che è sempre importante tenere presente, anche nell'epoca dei Giga- e dei Tera-Byte.

Quindi prima contiamo e creiamo il vettore e poi...??? A questo punto dobbiamo riavvolgere il nastro. Si esatto l'oggetto che abbiamo usato per leggere è arrivato alla fine del file e quindi non può più accedere ai dati. Bisogna dirgli di ricominciare da capo. Questo lo possiamo fare o con un trucco, cioè chiudiamo e riapriamo il file

```
leggimi.close();    //chiude il file
leggimi.open("dati.dat"); // la riapertura del file mette il "cursore"
                          // dell'ifstream all'inizio del file
```

oppure possiamo usare i metodi offerti dalla classe ifstream come segue

```
leggimi.clear(); //resetta le etichette di errore,
                 //è necessaria prima di una operazione di rewind
leggimi.seekg(0); //rimettiamo il riferimento del sistema
                  //di lettura al primo elemento, realizza il rewind
```

Ora siamo finalmente pronti per caricare i nostri dati in un vettore delle dimensioni giuste.

```
float misure[NumerodiLetture];//crea il vettore che fa per noi!
for(int i=0; i<NumerodiLetture; i++){
leggimi>>misure[i];
}
leggimi.close(); chiudiamo definitivamente!
```

I dati sono caricati in una variabile ed è a nostra completa disposizione, e quindi possiamo per esempio trovare qual'è il massimo e il minimo assoluto tra i dati che abbiamo letto:

```
//restituisce il valore massimo e il valore minimo
```

```
float max, min;
max=misure[0]; //inizializziamo al primo valore
min=misure[0];
for(i=0; i<NumerodiLetture; i++){
if(misure[i]>max)max=misure[i];
else if(misure[i]<min)min=misure[i];
}
cout<<"Il valore minimo e': "<<min<<endl;
cout<<"Il valore massimo e': "<<max<<endl;
```

Questo esercizio oltre a contribuire a renderci più familiare la scrittura di un codice, fornisce un pezzo di programma fondamentale per la costruzione di un istogramma.

3.6.3 Costruzione di un istogramma

Prima di iniziare a scrivere un programma per la risoluzione di un problema bisogna conoscere e capire bene il problema. Non avere nessun dubbio su ciò che ci serve e su cosa bisogna fare.

Quindi prima di tutto proviamo a capire che cosa sia un istogramma. Un istogramma è un modo "ordinato" di rappresentare i dati/misure ripetute di una stessa quantità (istogramma monodimensionale) o di più quantità in relazione tra di loro (istogramma multidimensionale) che ci consente di leggere in modo veloce le informazioni statistiche contenute nei dati.

Un modo di costruire un istogramma potrebbe essere quello di individuare il massimo e il minimo assoluto dell'insieme di dati che si vogliono usare per costruire l'istogramma; usare questi due valori per determinare l'intervallo di variabilità dell'istogramma; poi si divide l'intervallo di variabilità in un numero arbitrario di sotto-intervalli (di uguale ampiezza nel nostro esempio, ma esistono anche rappresentazioni con intervalli di ampiezza variabile), i sotto intervalli vengono chiamati in gergo "bin"[4]; infine bisogna contare per ogni sotto intervallo quante misure sono comprese nei rispettivi limiti del sotto intervallo stesso. Se in ogni sotto intervallo registriamo il numero di volte che una misura cade nel sotto intervallo, l'istogramma si dice delle frequenze assolute, se invece questo numero è diviso per il numero del campione allora si dice che l'istogramma è delle frequenze relative.

Di seguito viene riportato il codice capace di assolvere questo compito. Il codice stampa a schermo su una colonna il valore medio di ogni sotto-intervallo e su una seconda colonna la frequenza assoluta delle misure nel sotto-intervallo.

```
//creare istogramma
int rebinning=0;//entra comunque la prima volta
do{ //per ripetere il ciclo, e cambiare binnaggio
int nint;  //nint=numero intervalli
cout<<"\nInserisci il numero di intervalli dell'istogramma: \n";
cin>>nint;
float l=(max-min)/(float)nint;//larghezza intervalli
cout<<"larghezza intervalli: "<<l<<endl;
int bin[nint]; //crea un vettore di dimensione pari a nint
for(i=0;i<nint;i++)
bin[i]=0; //inizializza a zero i valori di ogni intervallo
```

[4] bin in inglese vuol dire secchio, parola azzeccata e figurativa in quanto i sotto intervalli di un istogramma si comportano esattamente come un contenitore nel quale uno accumula i dati come quando si mettono oggetti in un contenitore.

```
for(i=0;i<nint;i++)
for(int v=0; v<j; v++)
if((misure[v]>=min+i*l)&&(misure[v]<=min+(i+1)*l))
//per ogni intervallo conta quali valori cadono entro l'intervallo stesso
bin[i]++;
//incrementa il numero di valori per l'intervallo preso in considerazione
//cout << scientific;
cout << fixed; //risolve il problema dell'allineamento
cout<<"\nIstogramma:"<<endl;
for(int n=0; n<nint; n++)
cout<<min+l/2+n*l<<"\t"<<bin[n]<<endl;
//stampa istogramma e valori medi di ogni intervallino
cout<<"\n Numero diverso di intervalli? (Se si, premere 1):\n";
cin>>rebinning;
}while(rebinning==1);
```

Analizzando un file creato appositamente con un codice MC, una distribuzione gaussiana, con questo codice otteniamo:

```
Inserisci il numero di intervalli dell'istogramma:
15
Larghezza intervalli: 1.522647

Istogramma:
-4.049477 21
-2.526830 154
-1.004183 1038
0.518463 4618
2.041110 13894
3.563756 27787
5.086403 36796
6.609050 31854
8.131697 18555
9.654344 7240
11.176990 1821
12.699636 329
14.222283 40
15.744930 3
17.267576 0

Numero diverso di intervalli? (Se si, premere 1):
0
gmandaglio75@buccellato:>
```

Questa che vediamo è una rappresentazione numerica, non grafica, ma come dicevamo prima la tabella riassume delle informazioni statiche, infatti anche a colpo d'occhio notiamo che la massima frequenza è tra 5.086403 e 6.609050, quindi il valore medio della distribuzione sarà un numero compreso tra questi due valori. Infatti nel calcolo diretto del valore medio effettuato negli esempi precedenti è risultato essere pari a 5.35344.

In figura 3.2 è rappresentato l'istogramma computato con il nostro codice per 15 bin con Root e gnuplot. Più avanti vedremo che le librerie di Root forniscono delle classi che ci consentono di

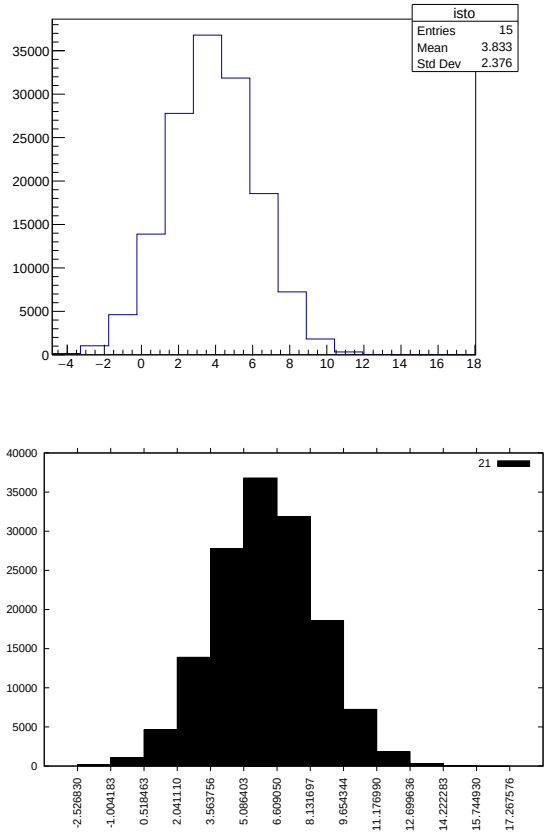


Figura 3.2: Rappresentazioni dell'istogramma computato per 15 divisioni con root in alto e gnuplot in basso.

costruire l'istogramma partendo dai dati grezzi. Il programma di grafica gnuplot ha bisogno che il lavoro di calcolo dell'istogramma sia fatto da noi. ROOT non è un programma di grafica ma un tool completo per l'analisi dati che possiede anche la capacità di fare grafica ad alto livello.

3.7 Funzioni

In c/c++ qualunque programma e sotto-programma sono funzioni. Anche il programma principale è una funzione e si chiama main (in inglese vuole dire principale). Tutte le funzioni, compresa la principale, hanno lo stesso paradigma: cosa restituisce - nome della funzione - tra parentesi tonde ciò che riceve in input - l'implementazione della funzione segue racchiusa tra graffe.

Il codice che segue è la semplice implementazione di una funzione che restituisce la somma di due numeri interi forniti alla funzione.

```
int somma(int a, int b){
return a+b;
}
```

Ora però vediamo come le funzioni vengono implementati all'interno di un programma, come vengono chiamate e che regole devono essere eseguite. Una regola che non ammette eccezioni è che ogni funzione, come del resto le variabili, deve essere dichiarata prima dell'esecuzione della funzione main. La dichiarazione della funzione si chiama prototipo. Il prototipo ripeto deve

essere sempre dichiarato, la dichiarazione può coincidere con l'implementazione oppure no. Le variabili che vengono usate per l'implementazione delle funzioni sono puramente simboliche, e in generale nella funzione principale non siamo tenuti a usare le stesse variabili. Le variabili dichiarate all'interno di una funzione sono variabili locali.

Esempio prototipo e implementazione prima della funzione principale:

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

int somma(int a, int b){
return a+b;
}
int main(){
int pino,pina;
pino =10;
pina=20;
cout<<somma(pino,pina)<<endl;
return 1;
}
```

Esempio prototipo prima della funzione principale e implementazione dopo la funzione principale:

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

int somma(int a, int b);

int main(){
int pino,pina;
pino =10;
pina=20;
cout<<somma(pino,pina)<<endl;
return 1;
}
int somma(int a, int b){
return a+b;
}
```

Esempio in cui la funzione accessoria somma è stata scritta in un file "header" somma.h:

```
#include <iostream>
#include "somma.h"
using namespace std;
int main(){
int pino,pina;
pino =10;
pina=20;
cout<<somma(pino,pina)<<endl;
return 1;
}
```

L'header file somma.h che in questo esempio vive nella stessa cartella che contiene il programma che lo carica attraverso la chiamata #include contiene prototipo e implementazione della funzione:

```
// file somma.h
int somma(int a, int b){
return a+b;
}
```

3.7.1 Passaggio di valori a funzioni via value o reference

Esercizi

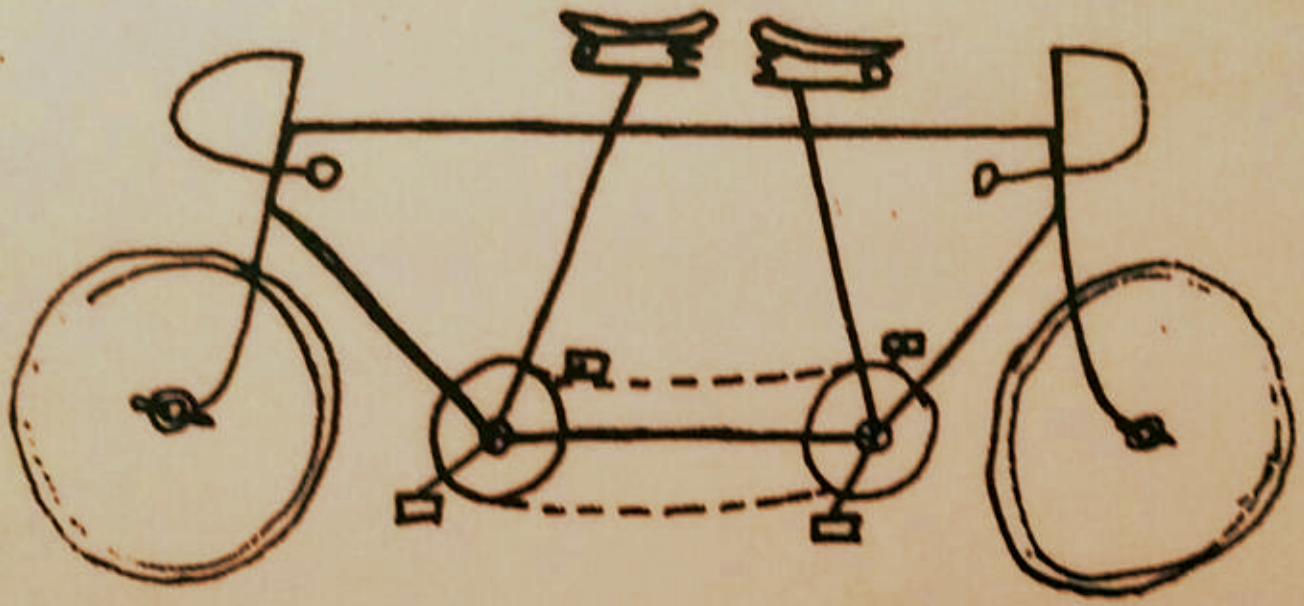
- usare una funzione che in input vuole una vettore di caratteri o una stringa che contenga il nome/o percorso+ nome di un file e in uscita fornisca il numero di valori scritti nel file.
- Scrivere una funzione che in input vuole una vettore di caratteri o una stringa che contenga il nome/o percorso+ nome di un file e in uscita contiene un vettore che contiene tutti i valori letti nel file.
- Scrivere una funzione che in input prenda un vettore di reali e che restituisca il massimo e il minimo assoluti contenuti dentro il vettore. Come possiamo avere due uscite invece che una?
- Scrivere una funzione che metta in evidenza l'utilità di utilizzare il passaggio di parametri per valore.

3.7.2 Ricorsività delle funzioni

Le funzioni hanno la proprietà di poter chiamare se stesse, questa proprietà viene detta ricorsività. Questa funzione è molto utile e può essere utilizzata per esempio quando si implementano dei codici di ricerca dicotomica (detta anche binaria), o per esempio se volessimo implementare l'operazione di fattoriale.

```
// calcolo del fattoriale
#include <iostream>
using namespace std;

long fattoriale (long valore)
{
  if (valore > 1)
   return (numero * fattoriale (valore-1));
  else
   return 1;
}

int main ()
{
  long Numero = 9;
  cout << Numero << "! = " << fattoriale (Numero);
  return 0;
}
```

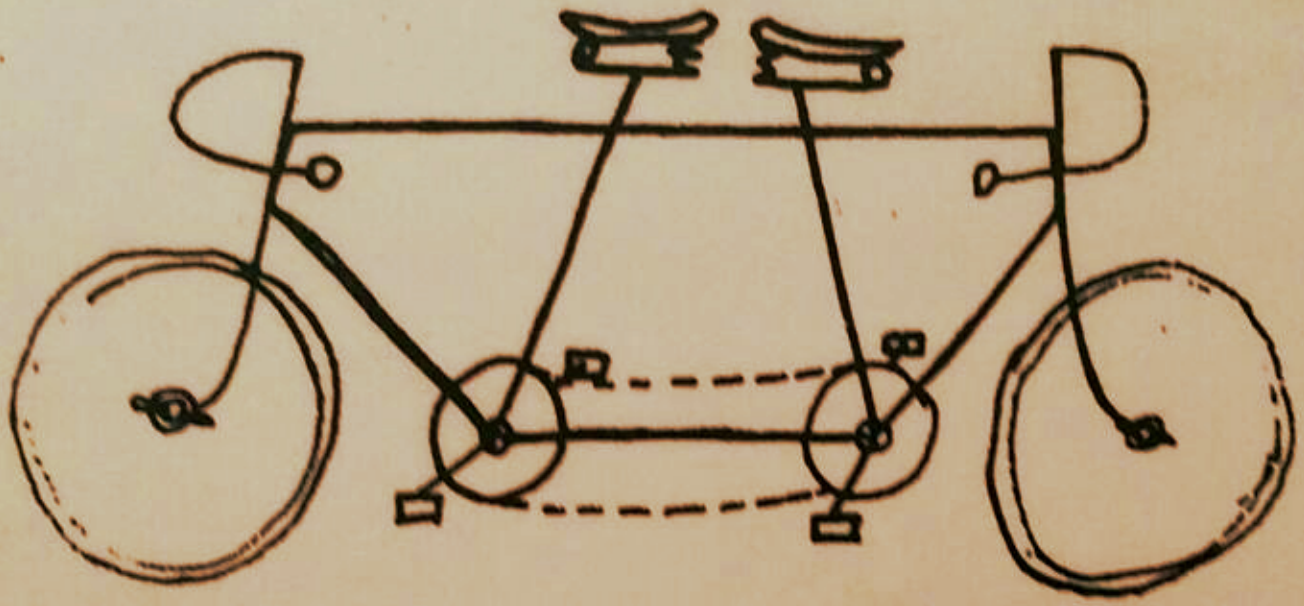
3.7.3 Template di funzione



4. Classi

PROBARE ET REPROBARE !

disegno di : Bruno Touschek



5. Dal C++ procedurale al Fortran

PROBARE ET REPROBARE !

disegno di : Bruno Touschek

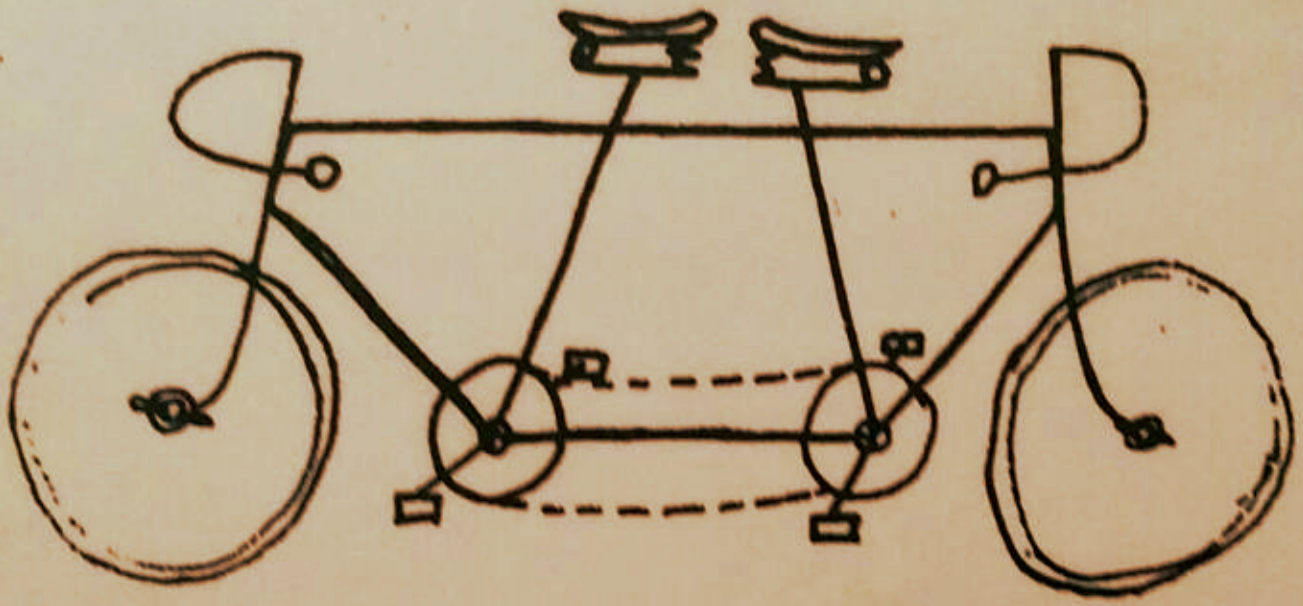
III Sturmenti di Analisi Dati

6 ROOT 45

7 Gnuplot 47

Bibliography 49
Books
Articles

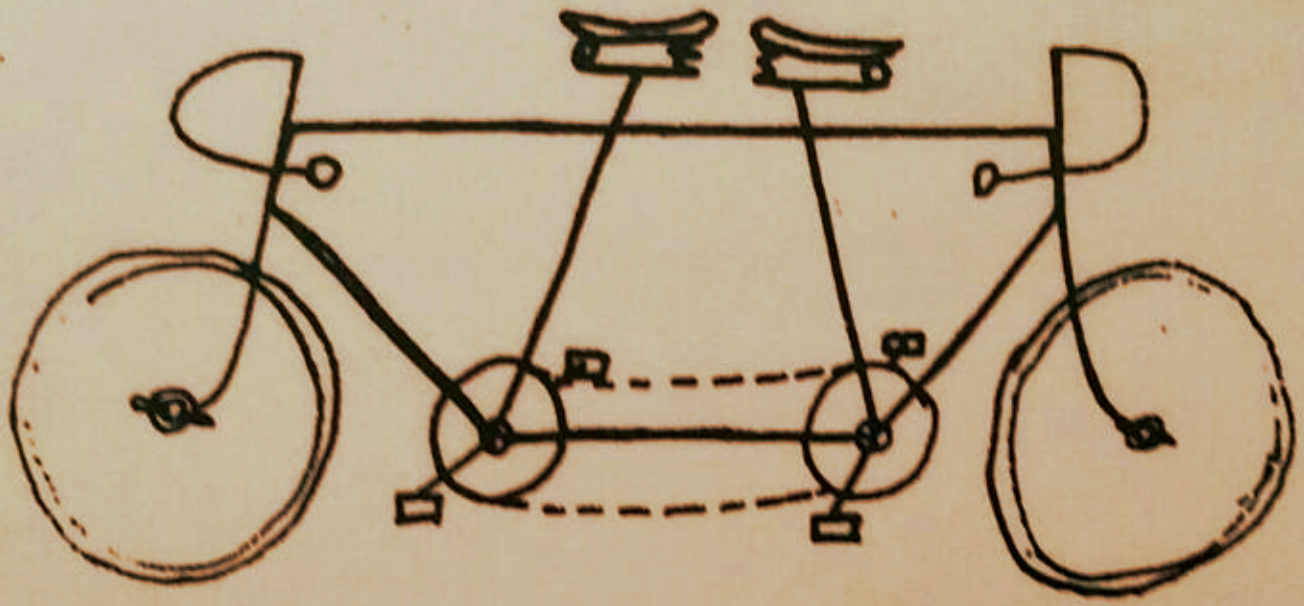
Index 51



6. ROOT

PROBARE ET REPROBARE !

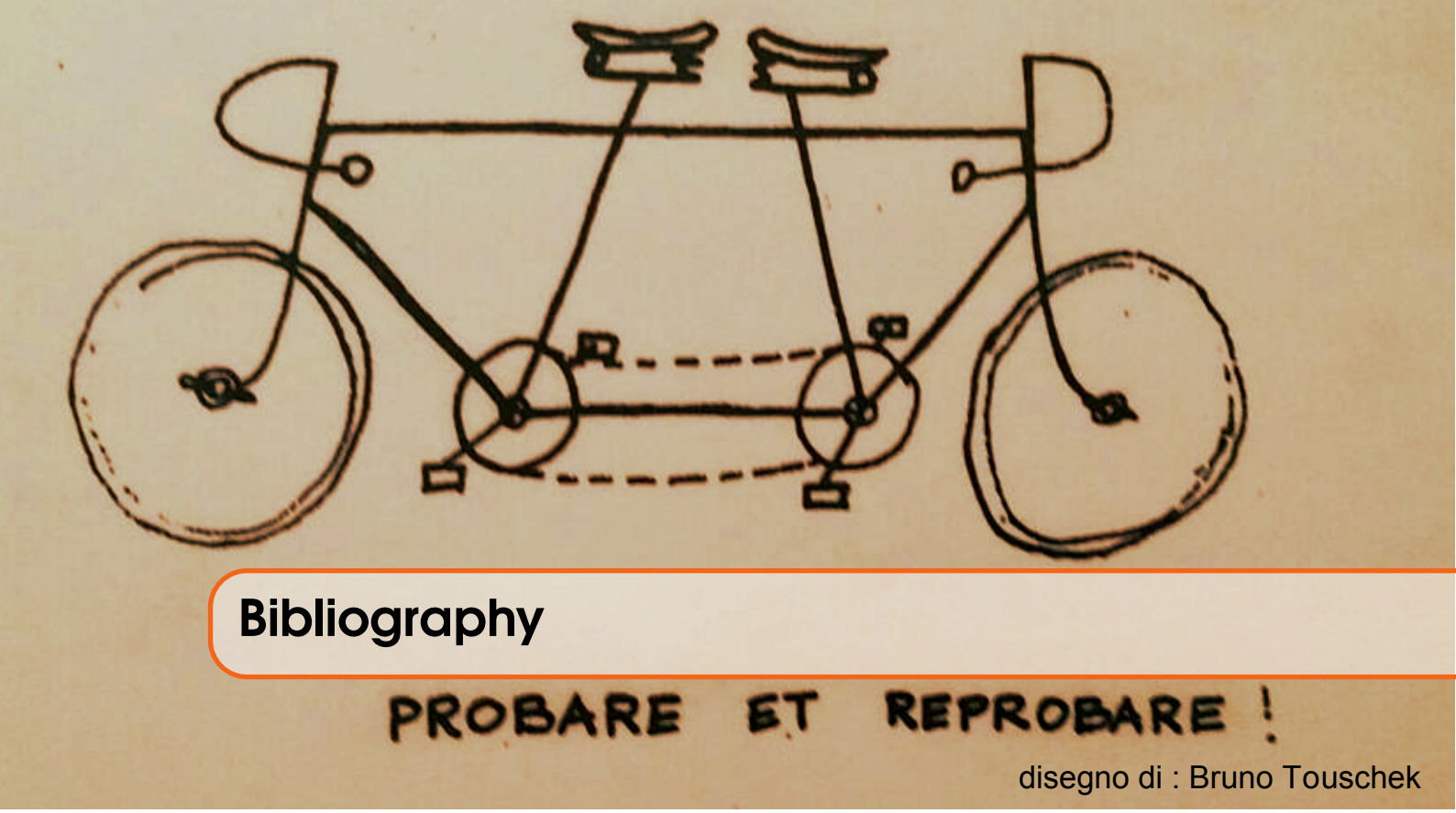
disegno di : Bruno Touschek



7. Gnuplot

PROBARE ET REPROBARE !

disegno di : Bruno Touschek

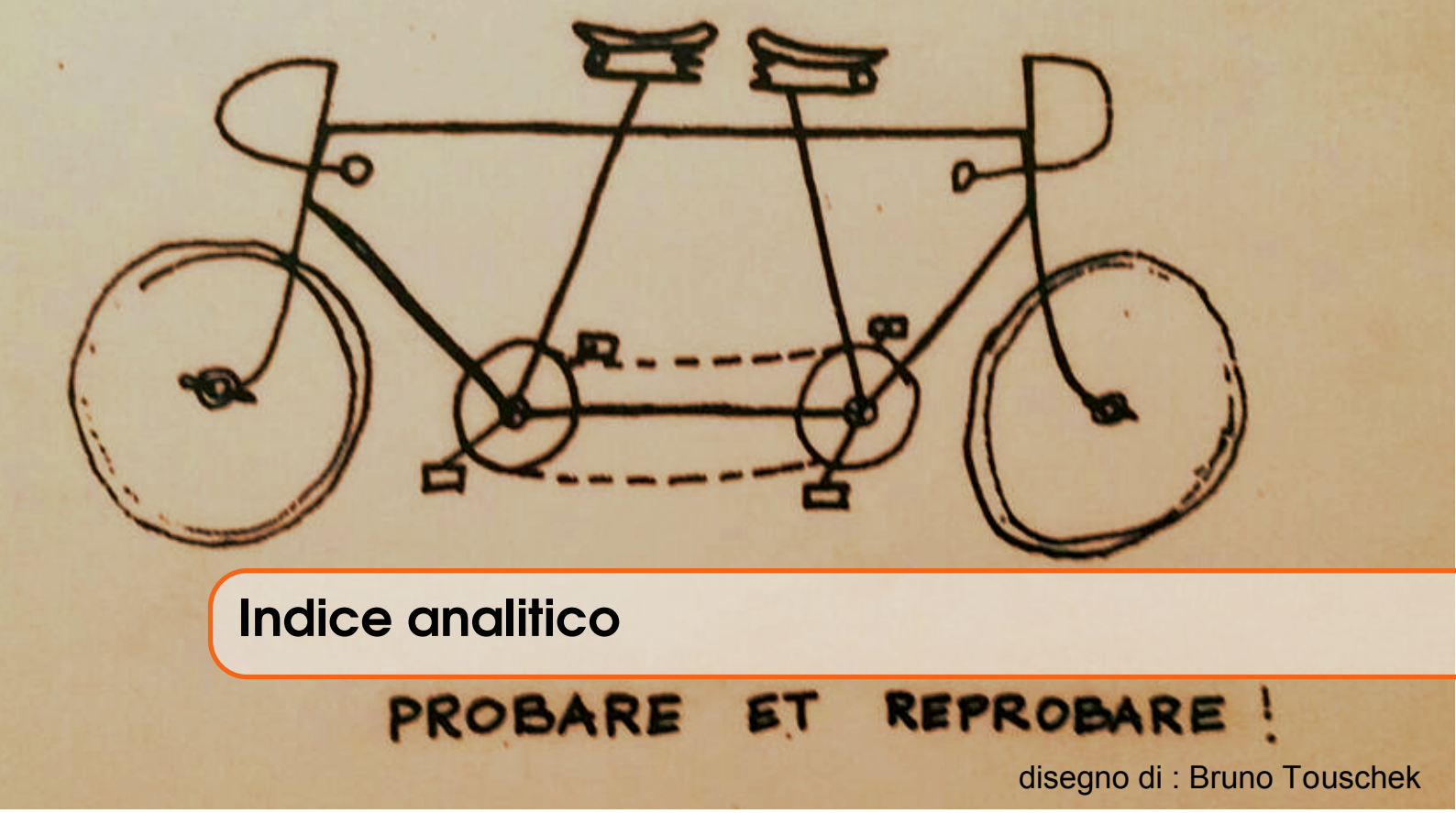


disegno di : Bruno Touschek

Bibliography

Books

Articles



disegno di : Bruno Touschek

Indice analitico

Citation, 8
Corollaries, 10

Definitions, 9

Examples, 10
    Equation and Text, 10
    Paragraph of Text, 11
Exercises, 11

Figure, 15

Lists, 8
    Bullet Points, 8
    Descriptions and Definitions, 8
    Numbered List, 8

Notations, 10

Paragraphs of Text, 7
Problems, 11
Propositions, 10
    Several Equations, 10
    Single Line, 10

Remarks, 10

Table, 15
Theorems, 9
    Several Equations, 9
    Single Line, 9

Vocabulary, 11