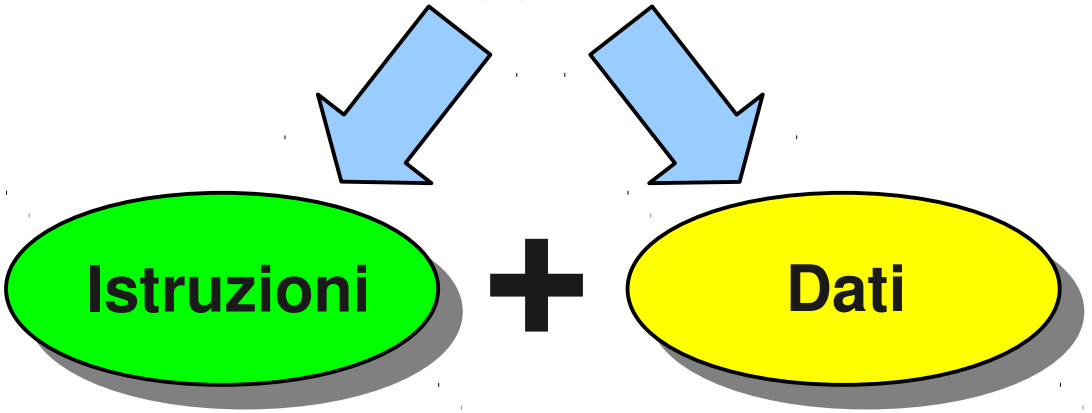
Programmazione ad oggetti

Programmazione ad oggetti

- Si **scompone** un problema nelle parti che lo costituiscono
- Ciascuna parte diventa un **oggetto**



- **Complessità** di programmazione notevolmente ridotta
- Ulteriori **vantaggi** dall'implementazione di:
  - Incapsulamento
  - Polimorfismo
  - Ereditarietà

Programmazione ad oggetti

Es: voglio accorpare le variabili e le funzionalità di un generico istogramma in una singola entità (**histo_t**) analoga ad un tipo predefinito (ad es. **int**).

```
int   var   (10) ;
```

Definisco un oggetto ***di tipo int*** contente il valore 10

tipo generico (con le sue funzioni: +, *, /, -...)

nome della variabile istanziata nel codice

valore di inizializzazione della variabile

Definisco un oggetto ***di tipo istogramma*** contente una particolare una distribuzione di dati descritta dai parametri

```
histo_t myHisto (/*parametri*/) ;
```

A differenza di un tipo predefinito, bisogna sviluppare la definizione del nuovo tipo, la **classe**.

Durante il programma, si istanziano gli **oggetti** di una **classe.**

Cos'è **(in pratica)** un oggetto

- un "tipo" **complesso**
  - contiene al suo interno tante variabili diverse
- un "tipo" **autosufficiente**
  - contiene le funzioni che servono per elaborare l'informazione che contiene
- un "tipo" **riservato**
  - i dati possono non essere direttamente accessibili dall'esterno

Programmazione ad oggetti

- Come si inventa una classe
  - la sua struttura generale
  - che cosa contiene
  - come si sviluppano le sue varie operazioni
  - le sue funzioni speciali

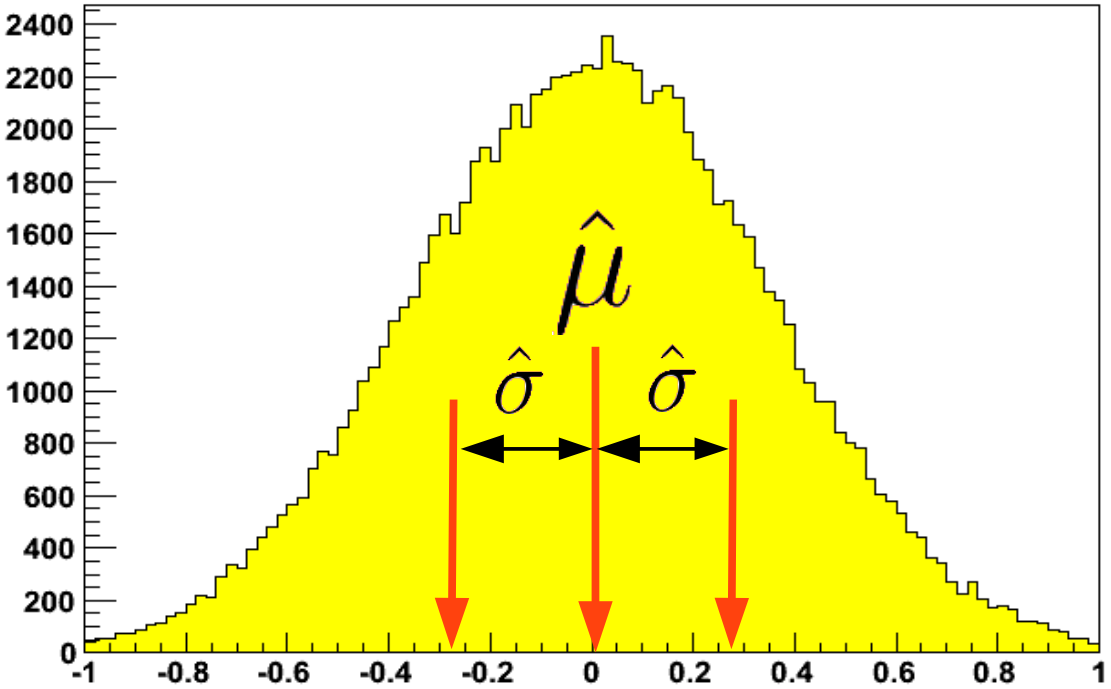
- Come si utilizzano gli oggetti
  - come cambia la programmazione, quando le funzioni vengono associate alle variabili all'interno delle classi

Perché gli istogrammi?

Un insieme di misure indipendenti può essere rappresentato da un istogramma.

Dopo aver raccolto un numero sufficiente di dati, l'istogramma sarà una buona rappresentazione della *pdf* (probability distribution function) dei dati.

Dall'istogramma si possono valutare le statistiche della distribuzione (media, varianza...)



L'istogramma "artigianale"

Abbiamo sviluppato un insieme di informazione e funzioni adatto a gestire una pdf discreta:

calcolo di statistiche

$$\sigma^2 = E[x^2] - \mu^2, \;\; E[x^2] = \frac{\sum x_i^2 \times N_i}{\sum N_i}$$

calcolo del massimo

```
int getHistoMax (int * histo, int nBins)
{
  int max = 0 ;
  for (int i = 0 ; i<nBins ; ++i)
    if (histo[i] > max) max = histo[i] ;
  return max ;
}
```

riempimento

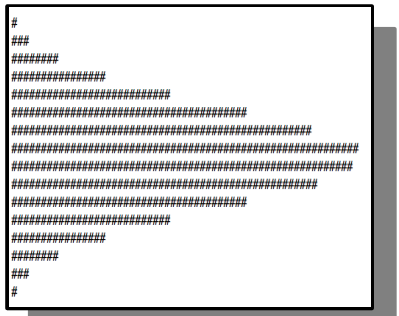
```
for (int it = 0; it < max_num; ++it)
 {
   double numero_casuale =
      rand_CLT (xMin, xMax, nTries) ;
   int index = floor ((numero_casuale - xMin)/step) ;
   ++histo[index] ;
 }
```

inizializzazione

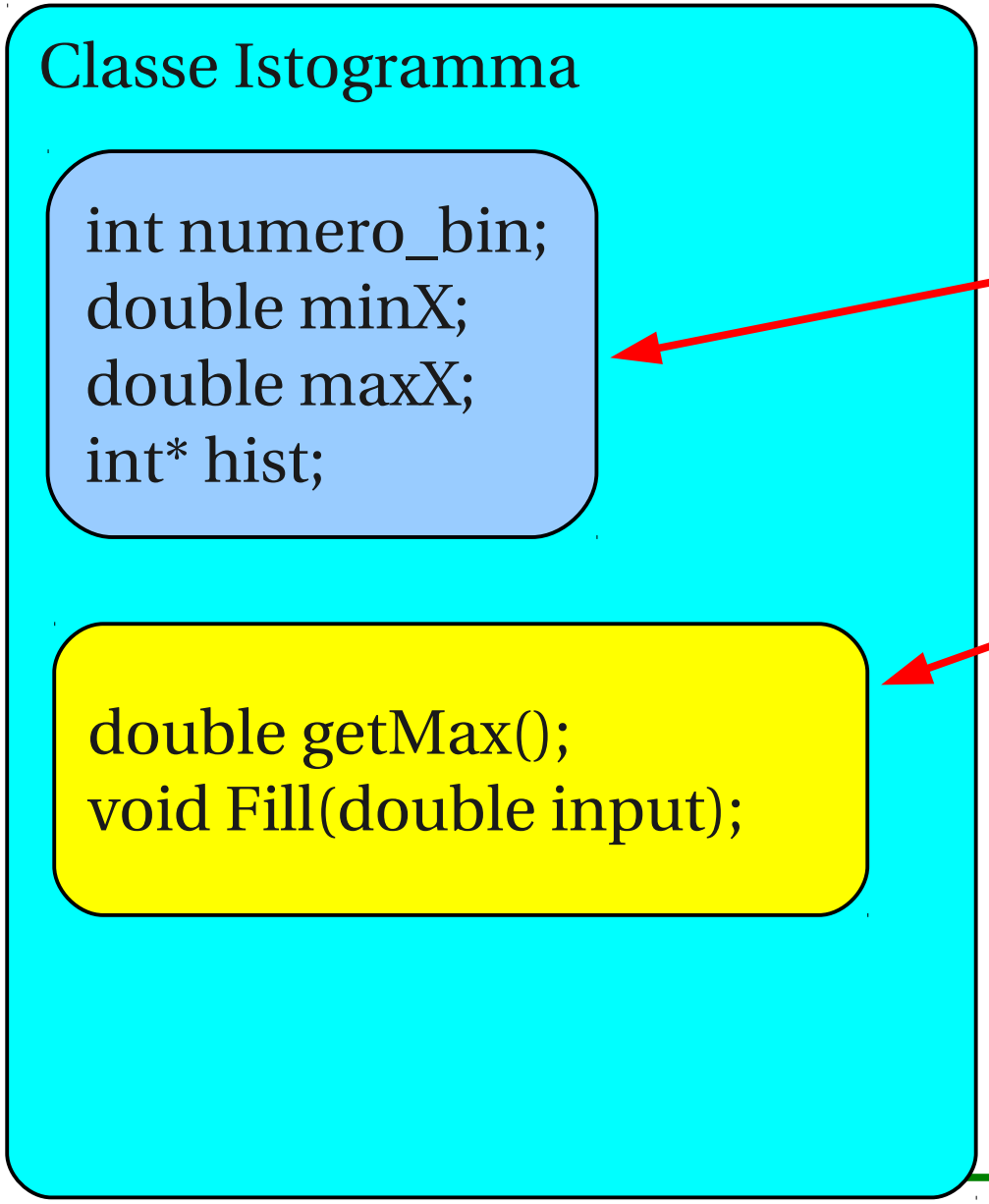
```
int nBins = 30 ;
int * histo = new int[nBins] ;
for (int i = 0 ; i<nBins ; ++i)
  histo[i] = 0 ;
```

si può mettere tutto insieme?

visualizzazione

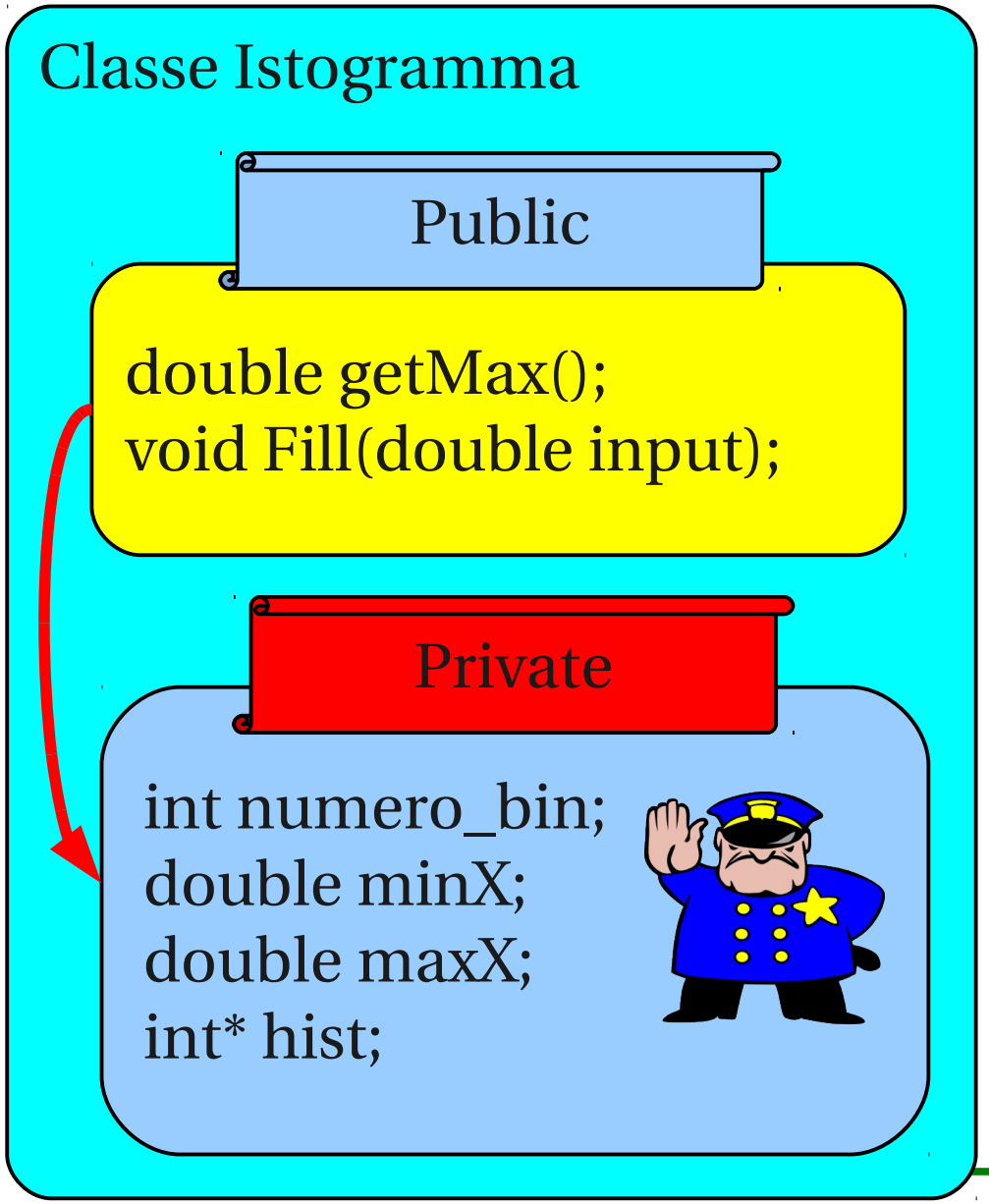


La classe istogramma



- All'interno di una classe ci sono i **membri:**
  - **Attributi**
    - e.g. Il numero di bin dell'istogramma, il range della variabile x, il vettore dei conteggi di ciascun bin...
  - **Metodi**
    - sono "funzioni" interne della classe;
      e.g. dimmi il bin con il numero di ingressi maggiore, riempi un bin con un entry, ...

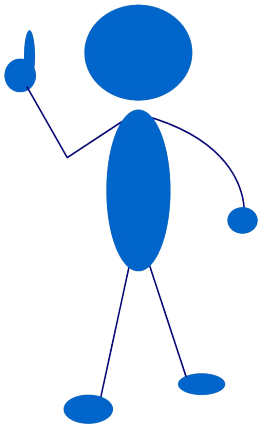
La classe istogramma



Di solito

Gli **attributi** sono nella parte "**privata**"

I **metodi** nella parte "**pubblica**"



Per avere maggior controllo, l'interazione con le variabili avviene tramite i metodi

Dove si scrive un oggetto

- Definito in un file ".h"
- Implementato in un file ".cc"
- Usato (istanziato) in un file ".cpp"

- Si compila con c++:

```
c++ classFile.cc programFile.cpp
-o exeFile
```

La classe istogramma: definizione

Definizione ***istogramma.h***

```
class istogramma
{
public:
  // default ctor
  istogramma ();
  // ctor
  istogramma (const int& nBin, const float& min, const float& max);
  // dtor
  ~istogramma ();

  //metodi
  int Fill (const float& value);
  void Print () const;
  float GetMean () const;

private:

  int nBin_p;
  float min_p;
  float max_p;
  float invStep_p;
  int* binContent_p;
  int entries_p;
  float sum_p;
};
```

class *nomeClasse*

public: *da quel punto inizia la zona "pubblica"*

private: *da quel punto inizia la zona "private"*

; *, il punto e virgola alla fine!*

La classe istogramma: definizione

Definizione ***istogramma.h***

```
class istogramma
{
public:
  // default ctor
  istogramma ();
  // ctor
  istogramma (const int& nBin, const float& min, const float& max);
  // dtor
  ~istogramma ();

  //metodi
  int Fill (const float& value);
  void Print () const;
  float GetMean () const;

private:

  int nBin_p;
  float min_p;
  float max_p;
  float invStep_p;
  int* binContent_p;
  int entries_p;
  float sum_p;
};
```

Costruttori : stesso nome della classe. È possibile costruire in modi diversi.

- Istogramma con range+numero di bin
- Istogramma con range+larghezza dei bin

Distruttori : ~ stesso nome della classe.

Metodi : definizione delle "funzioni" interne

Attributi : definizione degli "ingredienti" della classe

La classe istogramma: definizione


```
class istogramma
{
public:
..
..// default ctor
..istogramma ();
..// ctor
..istogramma (const int& nBin, const float&
..// dtor
..~istogramma ();
..
..//metodi
..int Fill (const float& value);
..void Print () const;
..float GetMean () const;
..
private:
..
..int nBin_p;
..float min_p;
..float max_p;
..float invStep_p;
..int* binContent_p;
..int entries_p;
..float sum_p;
};
```

Definizione ***istogramma.h***

Commentare!

// commento

/* commento */

Metodi : nomi delle funzioni riconoscibili

e.g.

Get... chiedere qualcosa alla classe

Set... imporre qualcosa alla classe

Fill... riempire qualche attributo

Attributi : nomi delle variabili riconoscibili

e.g. var_m (“member”)

var_p;

var_s;

La sua implementazione

Implementazione ***istogramma.cc***

```
istogramma::istogramma (const int &nBin, const float &min, const float &max):
 nBin_p (nBin),
 min_p (min),
 max_p (max),
 invStep_p ( nBin_p/(max_p-min_p) ),
 binContent_p ( new int[nBin_p] ),
 entries_p (0),
 sum_p (0.)
{
  //azzero gli elementi dell'istogramma
  for (int i = 0; i < nBin_p; ++i)
    binContent_p[i] = 0;
}


istogramma::~istogramma()
{
  delete[] binContent_p;
}
```

- **il costruttore**: dove si fa tutto quello che va necessariamente fatto per creare un oggetto

- **il distruttore**: dove si pulisce la memoria prima di eliminare l'oggetto per sempre

La sua implementazione

Implementazione ***istogramma.cc***

```
istogramma::istogramma (const int &nBin, const float &min, const float &max):
 nBin_p (nBin),
 min_p (min),
 max_p (max),
 invStep_p ( nBin_p/(max_p-min_p) ),
 binContent_p ( new int[nBin_p] ),
 entries_p (0),
 sum_p (0.)
{
  //azzero gli elementi dell'istogramma
  for (int i = 0; i < nBin_p; ++i)
    binContent_p[i] = 0;
}


istogramma::~istogramma()
{
  delete[] binContent_p;
}
```

nomeclasse::
Per definire costruttori, distruttore e metodi

Si possono inizializzare i membri in modo diverso:

Costruttore(valore1_init, vaore2_init) :
var1_p(valore1_init),
var2_p(valore2_init),

Oppure all'interno del costruttore
var1_p = valore1_init;
var2_p = valore2_init;

Il suo utilizzo

```
#include "istogramma.h"
int main()
{
  srand (time (NULL));

  //creo l'istogramma
  float min = 0.;
  float max = 1.;
  std::cout << "Inserisci gli estremi dell'istogramma [min,max): ";
  std::cin >> min >> max;

  int nBin = 0;
  std::cout << "Inserisci il numero di bin dell'istogramma: ";
  std::cin >> nBin;

  // ctor
  istogramma histo(nBin, min, max);


  // riempio l'istogramma
  histo.Fill(3.14);

  // stampo
  std::cout << "Mean = "<< std::setprecision(5) << histo.GetMean()<<std::endl;
  std::cout << "RMS  = "<< std::setprecision(5) << histo.GetRMS()<<std::endl;
  histo.Print();

  return 0;
}
```

Utilizzo ***test_istogramma.cpp***

- **includo il .h** dell'istogramma
- **creo l'istogamma**
  come un tipo *int numero (1);*
  qui viene chiamato il costruttore
- **riempio l'istogramma**
- **stampo l'istogramma**
- (il **distruttore** è chiamato automaticamente)

Tanti modi di istanziare...

```
int numero ;
int numero (5) ;
int altroNumero (numero) ;
int altroNumero = numero ;
```

... un tipo

con un valore di inizializzazione

con un altro oggetto dello stesso tipo

```
istogramma histo (0,2*M_PI,70) ;
istogramma histo2 (histo) ;
istogramma histo3 = histo ;
```

... un oggetto

per ogni modo di istanziare esiste un relativo costruttore

tutti i costruttori sono da scrivere!

Diversi costruttori

```
// default ctor
istogramma ();
// ctor
istogramma (const int& nBin, const float& min, const float& max);
// copy ctor
istogramma (const istogramma& original);
// operator =
istogramma& operator= (const istogramma& original);
```

costruttore

quando un oggetto viene istanziato con un valore di inizializzazione

l'argomento sono i valori di inizializzazione

copy ctor

quando viene istanziato da un altro oggetto della stessa classe

l'argomento è una *reference* ad un oggetto *const*

quando si definisce:
istogramma isto2 = isto1 ;
viene chiamato il copy ctor di isto2 !

operator=

quando viene ridefinito ponendolo uguale ad un altro oggetto della stessa classe

l'argomento è una *reference* ad un oggetto *const.*

Il copy ctor

```
istogramma::istogramma (const istogramma& original):
 nBin_p (original.nBin_p),
 min_p (original.min_p),
 max_p (original.max_p),
 invStep_p (original.invStep_p),
 binContent_p ( new int[original.nBin_p] ),
 entries_p (original.entries_p),
 sum_p (original.sum_p)
{
  //copio gli elementi dell'istogramma
  for (int i = 0; i < nBin_p; ++i)
    binContent_p[i] = original.binContent_p[i];
}
```

- dell'**oggetto `original`** è accessibile anche il *private*
- lista di **inizializzazione**
- **istruzioni interne**: per il vettore bisogna duplicare ogni singolo elemento!

L'operator= ()

```
istogramma& istogramma::operator= (const istogramma& original)
{
  if (binContent_p) delete[] binContent_p;

  nBin_p = original.nBin_p;
  min_p = original.min_p;
  max_p = original.max_p;
  invStep_p = original.invStep_p;
  binContent_p = new int[original.nBin_p];
  entries_p = original.entries_p;
  sum_p = original.sum_p;

  //copio gli elementi dell'istogramma
  for (int i = 0; i < nBin_p; ++i)
    binContent_p[i] = original.binContent_p[i];

  return *this;
}
```

- **l'oggetto `original`** è accessibile anche nei *private*
- lista di **inizializzazione**
- **istruzioni interne**: per ogni vettore bisogna duplicare ogni singolo elemento!
- la variabile **`this`** è un puntatore all'oggetto che la contiene l'operator= **restituisce l'oggetto** per poter fare:
  oggetto a = b = c = d ;

L'operator= ()

```
istogramma a;
istogramma b;
```

Costruisco i due oggetti
→ default constructor (devo averlo definito!)

```
a = b;
```

Viene chiamato l' Operator= ()
dell'oggetto "a"

L'oggetto "a" viene modificato
→ ritorno l'oggetto "a"

```
a = b = c;
```

c → b → a

Fill

```
int istogramma::Fill(const float& value)
{
  if (value < min_p)
  {
    ++underflow_p;
    return -1;
  }

  if (value >= max_p)
  {
    ++overflow_p;
    return -1;
  }

  else
  {
    ++entries_p;

    int bin = int( (value-min_p)*invStep_p );
    ++binContent_p[bin];

    sum_p += value;
    sum2_p += value*value;
    return bin;
  }

}
```

- Metodo per il riempimento dell´istogramma
- Condizioni per l´inserimento del valore
- Il valore del bin corrispondente viene aumentato

Print

```
void istogramma::Print() const
{
  // normalizza l'istogrammma al valore maggiore
  int max = 0;
  for (int i = 0; i < nBin_p; ++i)
  {
    if (binContent_p[i] > max) max = binContent_p[i] ;
  }

  // fattore di dilatazione per la rappresentazione dell'istogramma
  int scale = 50;

  // disegna l'asse y
  std::cout << "        +-------------------------------------------------------->"
<< std::endl;

  // disegna il contenuto dei bin
  for (int i = 0; i < nBin_p; ++i)
    {
      std::cout << std::fixed << std::setw(8) << std::setprecision(2)
<< min_p + i / invStep_p <<"|";
      int freq = int(scale * binContent_p[i] / max);
      for (int j = 0; j < freq; ++j)
        std::cout << "#";

      std::cout << std::endl;
    }
  std::cout << "        |\n" << std::endl;

}
```

Ricerca del massimo

Funzioni per la corretta visualizzazione (libreria <iomanip>)

Un esempio istogramma

Questo è l'output a schermo
nel caso di una distribuzione piatta

```
ndivara@pcepcma70:~/Desktop/Labo/lezione3/Lezione3$ ./testIstogramma1
Inserisci gli estremi dell'intervallo [a,b) in cui generare i numeri: 0 5
Inserisci quanti numeri casuali vuoi generare: 100000
Inserisci gli estremi dell'istogramma [min,max): 0 5
Inserisci il numero di bin dell'istogramma: 20
Mean = 2.5003
RMS  = 2.0832
overflow  = 0
underflow = 0
        +------------------------------------------------------------->
    0.00|##################################################
    0.25|##################################################
    0.50|##################################################
    0.75|##################################################
    1.00|##################################################
    1.25|##################################################
    1.50|##################################################
    1.75|##################################################
    2.00|##################################################
    2.25|##################################################
    2.50|##################################################
    2.75|##################################################
    3.00|##################################################
    3.25|##################################################
    3.50|##################################################
    3.75|##################################################
    4.00|##################################################
    4.25|##################################################
    4.50|##################################################
    4.75|##################################################
        |
```

const correctness

```
const int C1 = 10;
int const C1 = 10;

const int * C2;
int const * C2;

int * const C3;

int const * const C4;
```

C1: un intero il cui valore è costante

C2: un puntatore ad un “const int”, cioè un puntatore ad un intero costante

C3: un puntatore costante ad un intero variabile

C4: un puntatore costante ad un intero costante

const si applica alla prima cosa ci sia alla sua sinistra,
se non c'è nulla si applica alla prima cosa alla sua destra.

const in funzioni a classi

Funzioni

void funzione(int const &C1)

Passa alla funzione la referenza, ma non permette di modificare C1

Metodi e classi

classe::metodo() const;

Il metodo non può modificare nessun attributo della classe!

Se un oggetto è definito const si possono chiamare solo i metodi const dell'oggetto

I metodi “get” possono essere definiti “const”

Ancora un dettaglio

```
#ifndef istogramma_h
#define istogramma_h

/** \class istogramma
  semplice istogramma con strumetnto di visualizzazione
*/
class istogramma
{
  public:

   /*
     etc . . .
   */

  private:

   /*
     etc . . .
   */
} ;

#endif
```

istruzioni al precompilatore:

```
#ifndef
#define
#endif
```

per evitare di definire due volte lo stesso oggetto

Esercizio 1

- scrivere tutta la struttura
- implementare i metodi
- aggiungere i metodi che facciano:
  - **`getMean()`**
  - **`getRMS()`**
  - **`getIntegral()`**
  - **`getOverflow()`**
  - **`getUnderlow()`**

  (e quindi anche i membri necessari)

Esercizio 2

- Scrivere la classe dei numeri complessi

- Implementare i metodi:
  - **Re** () → parte reale
  - **Im**() → parte immaginaria
  - **Mod**() → modulo
  - **Rho**() e **Theta**() → descrizione polare di numeri complessi
  - **Print**() → Re + i Im

- Implementare gli operatori
  - =(), +(), -(), *(), /()
  - =(**const double& original**) // operazioni con double
  - ^(**const int& potenza**) // elevazione a potenza intera