AA 2024-2025

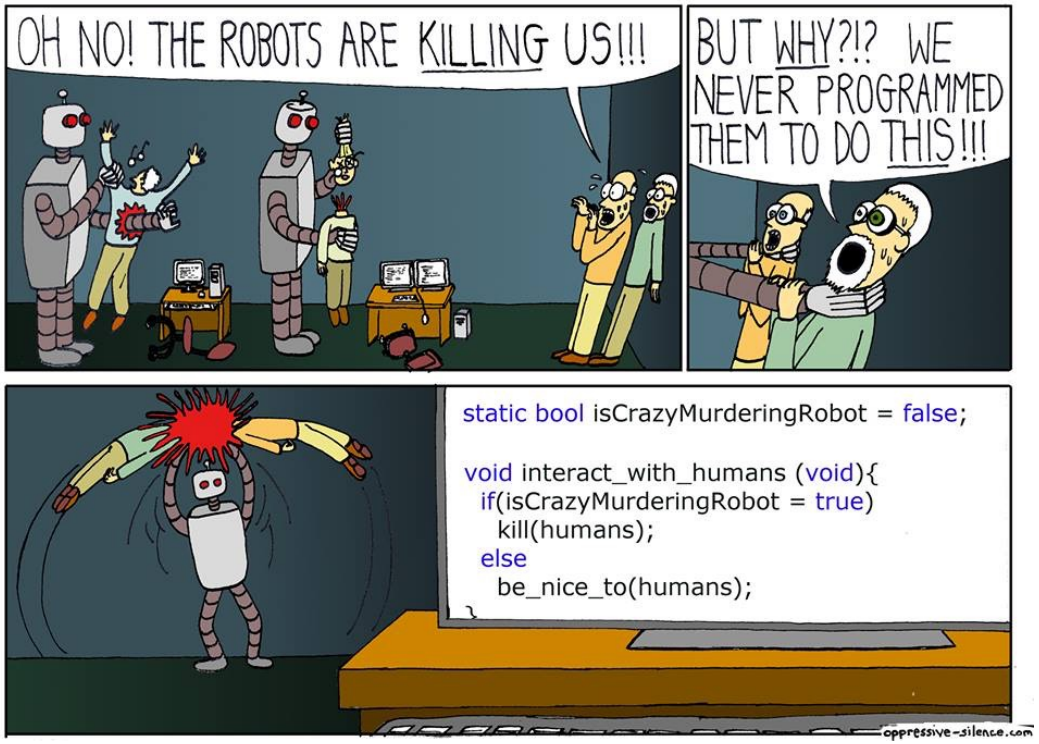
Lezione 1 : “Hello World”

Prof. L. Carminati
Università degli Studi di Milano

Laboratorio Trattamento Numerico dei Dati Sperimentali

Obiettivi del corso

- Approfondire l'uso di un **linguaggio di programmazione** "ricco" e completo (C++)
  - Riprendere elementi di base del linguaggio C : variabili, strutture di controllo, funzioni, algoritmi…
  - Introdurre elementi di C++ (programmazione ad oggetti) : classi, ereditarietà, polimorfismo.
  - Utilizzo librerie standard (contenitori e algoritmi)



- Utilizzare la programmazione per **risolvere problemi** : l'approccio numerico è molto spesso l'unica soluzione
  - Analisi di dati sperimentali (medie, varianze, istogrammi, correlazione)
  - Algoritmi : ricerca zeri, risoluzione numerica di integrali ed equazioni differenziali
  - Techiche Monte Carlo per simulazione di esperimenti e risoluzione di integrali

Dettaglio argomenti

Lezione	Argomento	Linguaggio
1	Analisi set di dati (media, varianza, mediana)	Basics, allocazione dinamica
2	Analisi set di dati (media, varianza, mediana)	Classi in C++
3	Analisi set di dati (media, varianza, mediana)	Contenitori STL e ROOT
4	Esempi di analisi dati	
5	Campo di dipolo / Campo gravitazionale	Ereditarieta' / incapsulamento
6	Ricerca zeri di una funzione	Polimorfismo
7	Quadratura numerica	
8	Equazioni differenziali: oscillatori armonici, moto in campo gravitazionale, moto particella carica in campo elettrico e magnetico	Overloading operatori matematici
9	Equazioni differenziali: oscillatori armonici, moto in campo gravitazionale, moto particella carica in campo elettrico e magnetico	
10	Generatori di numeri casuali, integrazione MC	
11	Generatori di numeri casuali, integrazione MC	
12	Simulazione di apparati sperimentali	

Dettaglio argomenti

	Quite boring
	Interesting
	Real fun

Lezione	Argomento	Linguaggio
1	Analisi set di dati (media, varianza, mediana)	Basics, allocazione dinamica
2	Analisi set di dati (media, varianza, mediana)	Classi in
3	Analisi set di dati (media, varianza, mediana)	e ROOT
4 (facoltativa)	Esempi di analisi dati	
5	Campo di dipolo / Campo gravitazionale	Ereditarieta' / incapsulamento
6	Ricerca zeri di una funzione	Polimorfismo
7	Quadratura numerica	
8	Equazioni differenziali: oscillatori armoni po gravitazionale, moto particella carica in campo elettrico	Overloading operatori matematici
9	Equazioni differenziali: oscil moto in campo gravitazionale, moto particella carica in cam agnetico	
10	Generatori di n ntegrazione MC	
11	Generatori di nu casuali, integrazione MC	
12	Simulazione di apparati sperimentali	

Dallo scorso anno

Dettaglio argomenti

	Quite boring
	Interesting
	Real fun

Lezione	Argomento	Linguaggio
1	Analisi set di dati (media, varianza, mediana)	Basics, allocazione dinamica
2	Analisi set di dati (media, varianza, mediana)	Classi in C++
3	Analisi set di dati (media, varianza, mediana)	Contenitori STL e ROOT
4	Esercizi / Analisi dati climatici	
5	Campo di dipolo / Campo gravitazionale	Ereditarieta' / incapsulamento
6	Ricerca zeri di una funzione	Polimorfismo
7	Quadratura numerica	
8	Equazioni differenziali: oscillatori armonici, moto in campo gravitazionale, moto particella carica in campo elettrico e magnetico	Overloading operatori matematici
9	Equazioni differenziali: oscillatori armonici, moto in campo gravitazionale, moto particella carica in campo elettrico e magnetico	
10	Generatori di numeri casuali, integrazione MC	
11	Generatori di numeri casuali, integrazione MC	
12	Simulazione di apparati sperimentali	

Calendario indicativo

		Teoria	Lunedi pomeriggio	Martedi pomeriggio	Mercoledi pomeriggio	Giovedi pomeriggio	Venerdi mattina
			T1	T2	T3	T4	T5
		Carminati	Carminati	Tomasi	Galli	Maino	Carminati
		12:30-14:30	14:30-17:30	14:30-17:30	14:30-17:30	14:30-17:30	8:30-11:30
Lezione 1	Arrays	23/09/2024	23/09/2024	24/09/2024	25/09/2024	26/09/2024	27/09/2024
Lezione 2	Vettore	30/09/2024	30/09/2024	01/10/2024	02/10/2024	03/10/2024	04/10/2024
Lezione 3	Template e vectors	07/10/2024	07/10/2024	08/10/2024	09/10/2024	10/10/2024	11/10/2024
Lezione 4	Analisi dati	14/10/2024	14/10/2024	15/10/2024	16/10/2024	17/10/2024	18/10/2024
Lezione 5	Classi ed ereditarieta	21/10/2024	21/10/2024	22/10/2024	23/10/2024	24/10/2024	25/10/2024
Lezione 6	Classi astratte e ricerca zeri	28/10/2024	28/10/2024	29/10/2024	30/10/2024	31/10/2024	08/11/2024
Lezione 7	Quadratura numerica	04/11/2024	04/11/2024	05/11/2024	06/11/2024	07/11/2024	15/11/2024
Lezione 8	Equazioni differenziali	11/11/2024	11/11/2024	12/11/2024	13/11/2024	14/11/2024	22/11/2024
Lezione 9	Equazioni differenziali	18/11/2024	18/11/2024	19/11/2024	20/11/2024	21/11/2024	29/11/2024
Lezione 10	Generatori numeri casuali	25/11/2024	25/11/2024	26/11/2024	27/11/2024	28/11/2024	06/12/2024
Lezione 11	Integrazione MC	02/12/2024	02/12/2024	03/12/2024	04/12/2024	05/12/2024	13/12/2024
Lezione 12	Metodi montecarlo	09/12/2024	09/12/2024	10/12/2024	11/12/2024	12/12/2024	20/12/2024

- Se non ci sono incidenti di percorso possiamo finire il laboratorio entro le vacanze di Natale
- I turni preliminari sono disponibili al seguente link: https://docs.google.com/spreadsheets/d/1002ZQgSrlOoVafaqkokW7Ckh7YBXcVj5vODCkpn-Epo/edit?usp=sharing
  - Non esitate a segnalarmi errori, inesattezze o dimenticanze

Technicalities

- L'insegnamento 2024/2025 si terrà in **presenza**
  - Lezione di teoria 12.45-14.15 (?) propedeutica alla sessione pratica di laboratorio. Verrà svolta in presenza in aula A (possible trasmissione sincrona).
  - Le sessioni di laboratorio NON avranno possibilità di connessione remota
  - In caso di problemi seri è possible (previo accordo con il docente) recuperare una sessione di laboratorio in un turno diverso da quello assegnato
- Riferimento sito ARIEL: comunicazioni inviate attraverso quel canale
  https://myariel.unimi.it/course/view.php?id=3553
- Lezioni guidate da tenere come riferimento per le sessioni di laboratorio
  - http://labmaster.mi.infn.it/Laboratorio2/labTNDS/
  - https://labtnds.docs.cern.ch/
- Ricevimento libero dopo lezione (anche prima o dopo le sessioni di lab) oppure su appuntamento per questioni più complesse
- Bibliografia verrà discussa a lezione: dispense e manuali standard di C++ (il libro di testo che avete va benissimo, ottime dispense in rete) e analisi statistica
  - Stackoverfflow la vostra luce in luoghi oscuri quando ogni altra luce si spegne

Organizzazione pratica

- Lezioni pratiche di programmazione:
  - 5 turni di laboratorio (il docente di riferimento per turno è indicativo), sincronizzati con quelli del laboratorio di fisica: inizio puntuale (no quarto d'ora accademico salvo diversi accordi con il docente di riferimento)
    - Lunedi : 14.30-17.30 (Carminati)
    - Martedi : 14.30-17.30 (Tomasi)
    - Mercoledi : 14.30-17.30 (Galli)
    - Giovedi : 14.30-17.30 (Maino)
    - Venerdi : 8.30-11.30 (Carminati)
  - 12 lezioni da 3 ore.
  - Materiale didattico necessario sul sito web del laboratorio.
  - Esercizi svolti in parte in modo guidato, in parte autonomo
  - Si può programmare dalle macchine del laboratorio o dal proprio PC personale
- La presenza al laboratorio è obbligatoria : verranno controllate le presenze (!)

Technicalities

- Non ci sono propedeuticità rispetto al corso di informatica.
  - L'inizio dovrebbe essere abbastanza soft, C procedurale
  - Si assume però che si conoscano i tipi di variabile e le principali strutture di controllo

- Modalità esame :
  1. Consegna degli esercizi (programmi) assegnati durante il laboratorio
  2. Esercizio di programmazione in laboratorio (da svolgere singolarmente)
  3. Esame orale :
     - Discussione linea per linea di un vostro codice, discussione caratteristiche del linguaggio, costruzione di un programma al volo
     - A partire da un vostro codice discutere l'algoritmo implementato e le sue proprieta' e in generale gli argomenti sviluppati nelle lezioni teoriche

Esempio di test finale

Funzione integrale è un for() loop : fisso un valore di x e calcolo l'integrale.

Esercizio 1

L'ampiezza della figura di diffrazione generata da luce di lunghezza d'onda λ attraverso una fenditura di apertura d=80 μm nella posizione x su uno schermo distante L=1 m è data da un integrale del tipo:

$$A(x) = \int_{-d/2}^{d/2} dt \frac{1}{d} \cos\left[\frac{L}{\lambda}\left(\sqrt{1+\left(\frac{x-t}{L}\right)^2} - \sqrt{1+\frac{x}{L^2}}^{\,2}\right)\right]$$

1. Data una lunghezza d'onda di λ=500 nm fare un grafico di $A(x)$ per -10 cm < x < 10 cm.
2. Calcolare il valore più basso di $|x|$ per cui l'ampiezza è nulla.
3. Ripetere il punto precedente per λ=400 nm e 450 nm.

Calcolare gli integrali con il metodo dei trapezoidi ed un errore assoluto di almeno 10^{-4}.

Esercizio 2

Si implementi una classe in grado di generare numeri casuali distribuiti secondo la densità di probabilità $g(x) = \frac{1}{2}\sin x$ con x nell'intervallo [0,π]:

1. con il metodo *accept-reject*;
2. con il metodo della trasformata.

Quest'ultimo metodo si basa sul fatto che se r è una variabile casuale distribuita uniformemente in [0,1] e data la funzione $F(\xi) = \int_{x_{\min}}^{\xi} g(t)dt$, allora $x = F^{-1}(r)$ segue la distribuzione $g(x)$.

In entrambi i casi, produrre un istogramma di valori generati.

Esempio di test finale

Funzione integrale e' un for() loop : fisso un valore di x e calcolo l'integrale.

Esercizio 1

L'ampiezza della figura di diffrazione generata da luce di lunghezza d'onda λ attraverso una fenditura di apertura d=80 μm nella posizione x su uno schermo distante L=1 m è data da un integrale del tipo:

$$A(x) = \int_{-d/2}^{d/2} dt \frac{1}{d} \cos\left[\frac{L}{\lambda}\left(\sqrt{1+\left(\frac{x-t}{L}\right)^2} - \sqrt{1+\frac{x}{L^2}}^{\,2}\right)\right]$$

Sappiamo bene che è faticoso per via del carico generale di lavoro che avrete ma cercate di seguire regolarmente, terminate le esercitazioni a casa, tartassare docenti/assistenti e fare subito l'esame !

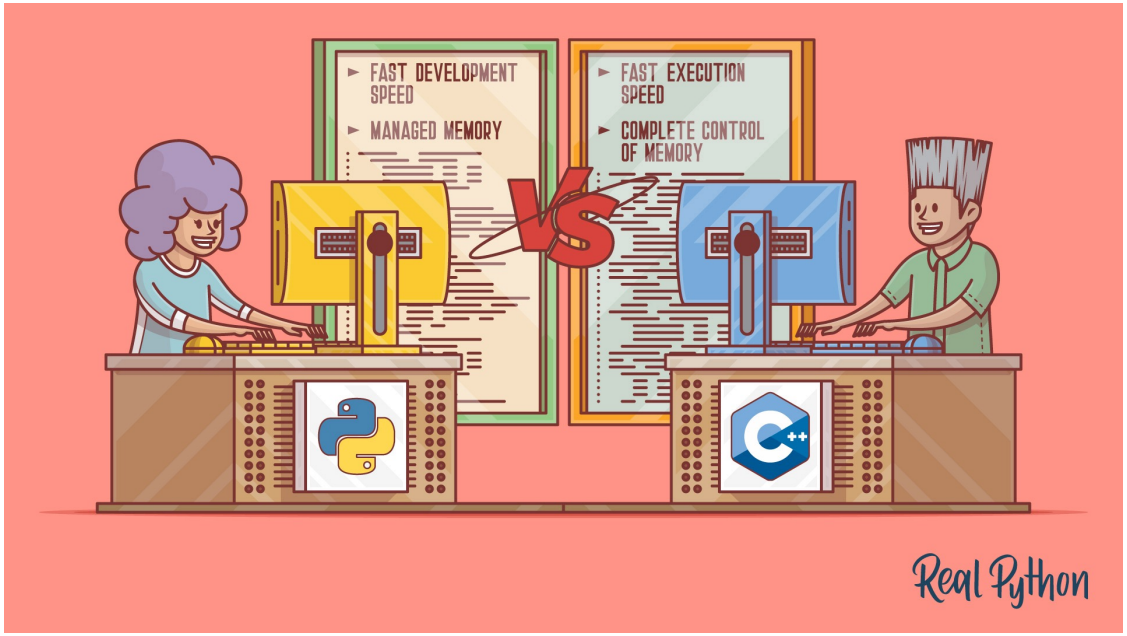
probabilità $g(x) = \frac{1}{2}\sin x$ con x nell'intervallo $[0,\pi]$:

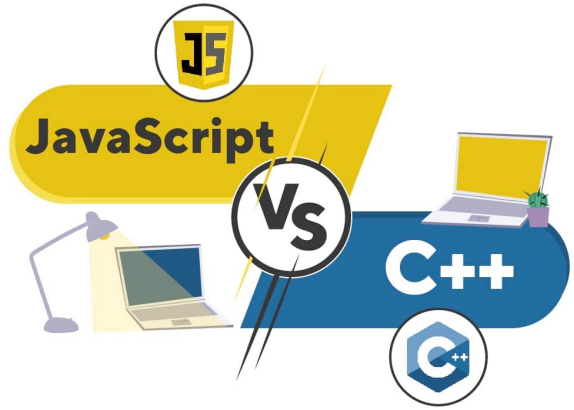
1. con il metodo *accept-reject*;
2. con il metodo della trasformata.

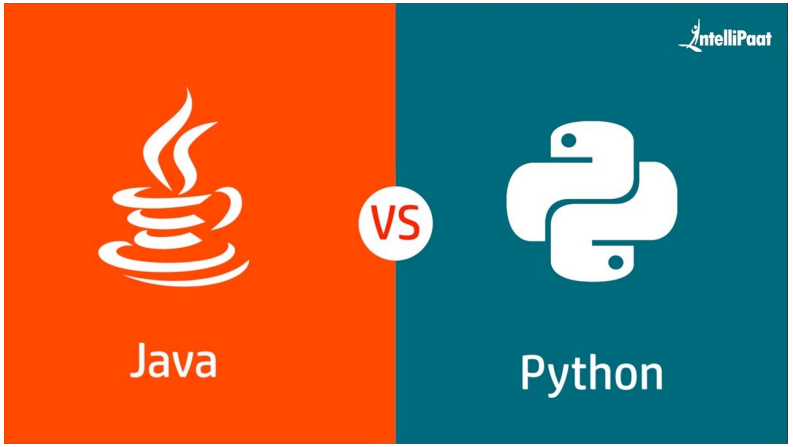
Quest'ultimo metodo si basa sul fatto che se r è una variabile casuale distribuita uniformemente in $[0,1]$ e data la funzione $F(\xi) = \int_{x_{\min}}^{\xi} g(t)dt$, allora $x = F^{-1}(r)$ segue la distribuzione $g(x)$.

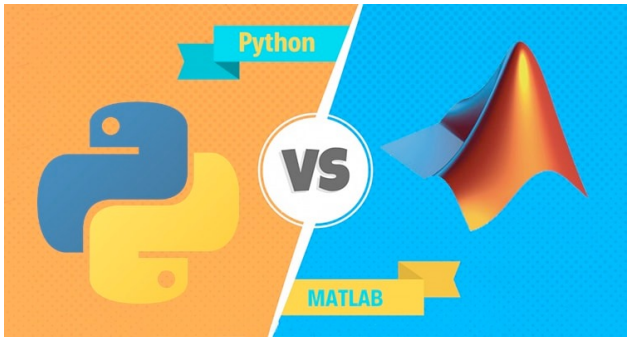
In entrambi i casi, produrre un istogramma di valori generati.

Obiettivi del corso : perche' C++ ?









Choose the right tool for your job !

Motivation: simple example of climate data analysis

- Evaluate the evolution with time of major climatic indicators (eg. Temperature)
- Use ERA5 re-analysis data :
  - "Reanalysis combines model data with observations from across the world into a globally complete and consistent dataset using the laws of physics. This principle, called data assimilation, is based on the method used by numerical weather prediction centres, where every so many hours (12 hours at ECMWF) a previous forecast is combined with newly available observations in an optimal way to produce a new best estimate of the state of the atmosphere, called analysis, from which an updated, improved forecast is issued. Reanalysis works in the same way, but at reduced resolution to allow for the provision of a dataset spanning back several decades. Reanalysis does not have the constraint of issuing timely forecasts, so there is more time to collect observations, and when going further back in time, to allow for the ingestion of improved versions of the original observations, which all benefit the quality of the reanalysis product"
- Average temperature of air at 2m above the surface available for each day since 1940.
- Data has been re-gridded to a regular (lat,lon) grid of 0.25 degrees for the reanalysis
- Reference portal : https://open-meteo.com/

Motivation : simple example of climate data analysis

- Goal of the analysis : estimate the evolution of the temperature of air at 2m above the surface in the Milano area from 1941 to 2023.

- What we have done :
  - For each day of the year compute the average temperature in the 1941-2023 time-range
  - For each day of the year compute the difference (Δ) between the actual value of the temperature and the average over 1941-2023 time-range in this day.
  - For each year we produced a file with the Δs in each day (a column with 365 entries)

- What you will find :
  - Total of 82 files (eg "1941.txt")

- What you will do :
  - Compute average Δs (+ uncertainties) for each year and : if no climate change, we would expect a gaussian peaked at 0
  - Draw the evolution of the average deltas with time (+ uncertainty band)

Lesson 1 : data analysis with C-like arrays

https://labtnds.docs.cern.ch/Lezione1/Lezione1/

- Data type : simple `double` numbers
- Data container type : dynamic C-like array
- Operations on data through functions

Straight to the point (no-frills) : my first program

```
gedit hw1.cpp &
```

1. Open your favourite editor

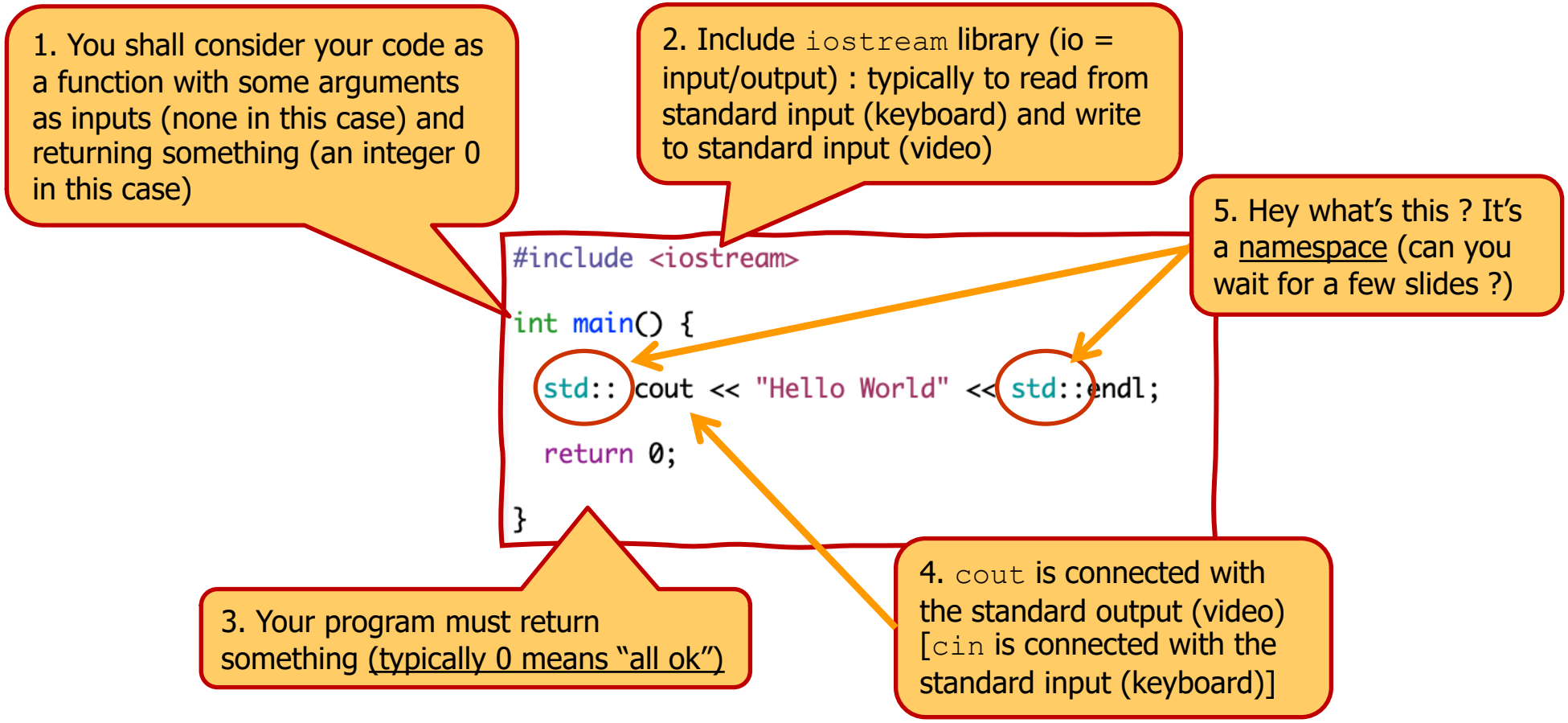
2. Write your code

3. Compile your code invoking g++

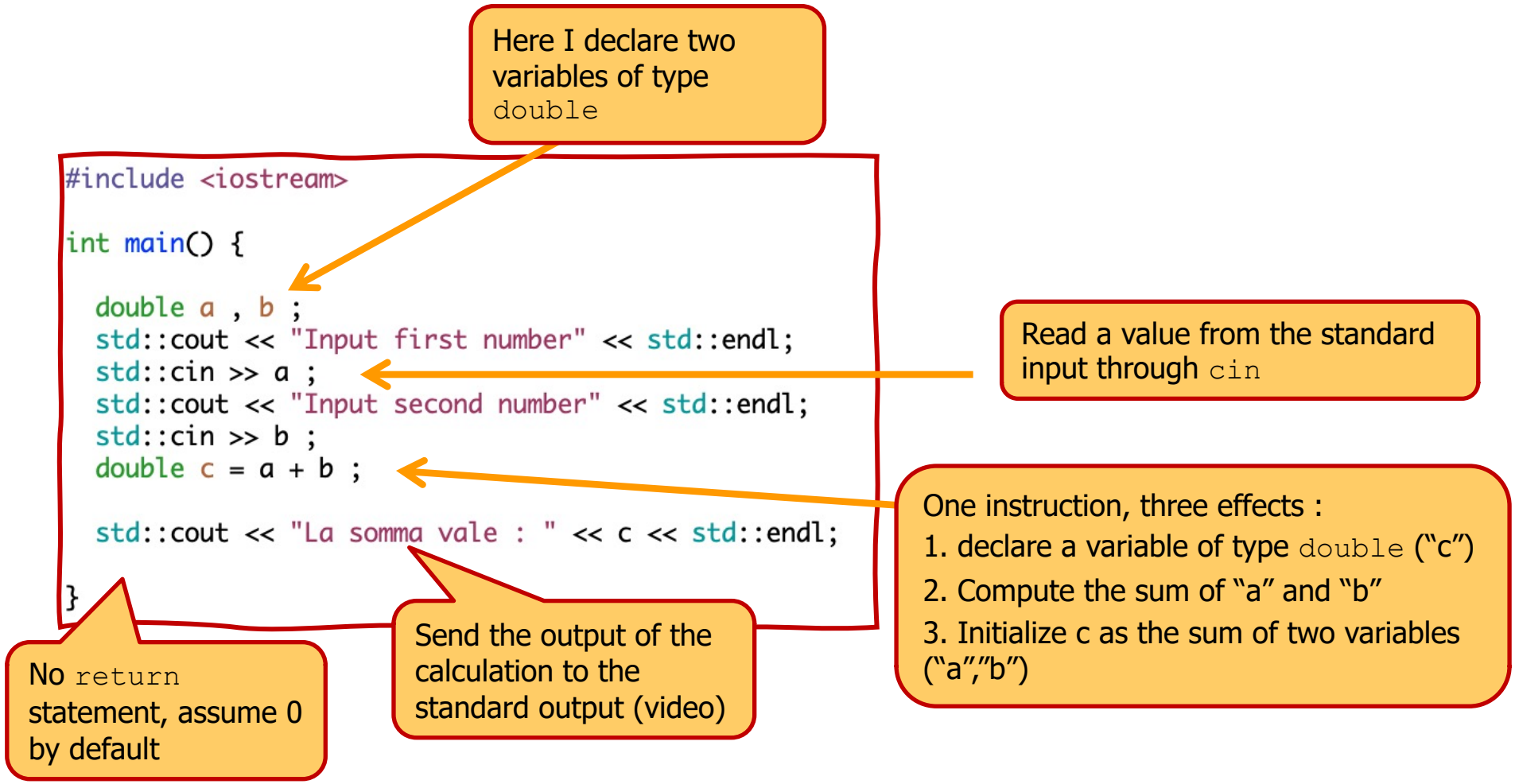
```
g++ hw1.cpp -o hw1
./hw1
Hello world
```

4. Execute your code

1. My first program : details



2. Standard input/output



3. Variables types

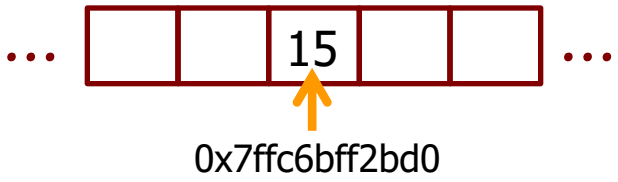
```cpp
#include <iostream>

int main() {

  int n ;
  unsigned int k = 0;
  double a = 1;
  double b = 2;
  double c = a + b ;

  double * address = &a ;
  std::cout << address << " " << *address << std::endl;

  char nome[10] = "foo" ;

  std:: cout << "La somma vale : " << c << std::endl;
  std:: cout << "Il contenuto della variabile nome : " << nome << std::endl;

}
```

A few variables types

OMG pointers !

`nome` can contain up to 10 characters

3. Variables types : pointers



- `(double *)` : this is the variable type. "address of A" is a variable of type `(double *)`, a variable built to host an address to double
- `&a` : & is the reference operator, returns the address of a variable

The address of variable "a" (&a) is deposited into an address variable called "address"

```
double a = 15 ;

double * address = & a ;

std::cout << "Valore contenuto in a          " << a << std::endl;
std::cout << "L'indirizzo della variabile a " << address << std::endl;
std::cout << "Valore contenuto in a          " << *address << std::endl;
```

de-referentiation operator (*) : when applied to a pointer returns the content

```
Valore contenuto in a          15
L'indirizzo della variabile a 0x7ffc6bff2bd0
Valore contenuto in a          15
```

- & (variable) returns the address of a variable
- * (pointer) returns the variable pointed by the address

4. Control structures

```
#include <iostream>

using namespace std;

int main() {

  int k = 0;

  if ( k < 10 ) {
    cout << "k is smaller than 10" << endl;
  } else {
    cout << "k is larger than 10" << endl;
  };

  for ( int k = 0 ; k < 10 ; k++ ) {
    cout << "k = " << k << end;
  };

  while ( k < 10 ) {
    cout << "k = " << k << end;
    k++;
  }

}
```

Selection statement : if (condition) {} else {}

Iteration statement : the for loop

Iteration statement : the while loop (check also also the do{} while () flavor)

5. Functions : structuring a bit the code

Declare functions :

<return_type> <function name> (<input1>,<input2>..)

```
#include <iostream>

double sum ( double a, double b) {
  return a+b;
};

int main() {

  double a , b ;
  std::cout << "Input first number" << std::endl;
  std::cin >> a ;
  std::cout << "Input second number" << std::endl;
  std::cin >> b ;
  double c = sum(a,b) ;
  std:: cout << "La somma vale : " << c << std::endl;

}
```

Call your function : sum returns a double which is deposited into the variable c

Organising the code in functions is crucial: always create functions when possible !

- ❑ Simplify the main program, a function can be called several times: never duplicate code !
- ❑ A function can be saved in a separate file and re-used in different projects

5. Functions : connections with the main

```
#include <iostream>

void scambiaByValue( double a, double b) {
  double temp = a ;
  a = b;
  b = temp;
}

void scambiaByReference( double &a , double &b ) {
  double temp = a ;
  a = b;
  b = temp;
}

void scambiaByPointer( double* a , double* b) {
  double c=*a;
  *a=*b;
  *b=c;
}

int main() {
  double a = 5 ;
  double b = 4 ;
  std:: cout << a << " " << c << std::endl;
  scambiaByValue(a,b) ;
  std:: cout << a << " " << c << std::endl;
  scambiaByReference(a,b) ;
  std:: cout << a << " " << c << std::endl;
  scambiaByPointer(&a,&b) ;
}
```

Let's write functions to exchange the content of two variables.

- Functions can be placed before the main

5. Functions : connections with the main

```
#include <iostream>

void scambiaByValue( double a, double b) {
  double temp = a ;
  a = b;
  b = temp;
}

void scambiaByReference( double &a , double &b ) {
  double temp = a ;
  a = b;
  b = temp;
}

void scambiaByPointer( double* a , double* b) {
  double c=*a;
  *a=*b;
  *b=c;
}

int main() {
  double a = 5 ;
  double b = 4 ;
  std:: cout << a << " " << c << std::endl;
  scambiaByValue(a,b) ;
  std:: cout << a << " " << c << std::endl;
  scambiaByReference(a,b) ;
  std:: cout << a << " " << c << std::endl;
  scambiaByPointer(&a,&b) ;
}
```

Let's write functions to exchange the content of two variables.

- Functions can be placed before the main

• Passing by value : when the function is called a local copy of a and b is built (no relation with the variables a and b in the main).

- Effect on a and b variables in the function
- No effect on the variables a and b in the main

5. Functions : connections with the main

```
#include <iostream>

void scambiaByValue( double a, double b) {
  double temp = a ;
  a = b;
  b = temp;
}

void scambiaByReference( double &a , double &b ) {
  double temp = a ;
  a = b;
  b = temp;
}

void scambiaByPointer( double* a , double* b) {
  double c=*a;
  *a=*b;
  *b=c;
}

int main() {
  double a = 5 ;
  double b = 4 ;
  std:: cout << a << " " << c << std::endl;
  scambiaByValue(a,b) ;
  std:: cout << a << " " << c << std::endl;
  scambiaByReference(a,b) ;
  std:: cout << a << " " << c << std::endl;
  scambiaByPointer(&a,&b) ;
}
```

Let's write functions to exchange the content of two variables.

- Functions can be placed before the main

• Passing by pointer: you pass the addresses of a and b in the main. The function works on addresses so the modifications in the function acts on the variables in the main

- Variables a and b in the main are exchanged

5. Functions : connections with the main

```
#include <iostream>

void scambiaByValue( double a, double b) {
  double temp = a ;
  a = b;
  b = temp;
}

void scambiaByReference( double &a , double &b ) {
  double temp = a ;
  a = b;
  b = temp;
}

void scambiaByPointer( double* a , double* b) {
  double c=*a;
  *a=*b;
  *b=c;
}

int main() {
  double a = 5 ;
  double b = 4 ;
  std:: cout << a << " " << c << std::endl;
  scambiaByValue(a,b) ;
  std:: cout << a << " " << c << std::endl;
  scambiaByReference(a,b) ;
  std:: cout << a << " " << c << std::endl;
  scambiaByPointer(&a,&b) ;
}
```

Let's write functions to exchange the content of two variables.

- Functions can be placed before the main

• Passing by reference: the magic & tells the compiler to consider the variable in the main. In this case a and b inside the function are exactly the a and b in the main

- Variables a and b in the main are exchanged
- Works as for pointers but the syntax is easier !

Intermezzo (1) : variables scope

https://cplusplus.com/

Named entities, such as variables, functions, and compound types need to be declared before being used in C++. The point in the program where this declaration happens influences its visibility:

- An entity declared outside any block has *global scope*, meaning that its name is valid anywhere in the code (*global variables*).
- While an entity declared within a block, such as a function or a selective statement, has *block scope*, and is only visible within the specific block in which it is declared, but not outside it (*local variables*).

Intermezzo (1) : variables scope

https://cplusplus.com/

Global variable

```cpp
int glob;         // global variable

int some_function ()
{
  int loc;        // local variable
  loc = 0;
}

int other_function ()
{
  glob = 1;  // ok: glob is a global variable
  loc  = 2;  // wrong: loc is not visible from this function
}

int main ()
{
  glob = 1;  // ok: glob is a global variable
  loc  = 2;  // wrong: loc is not visible in the main
}
```

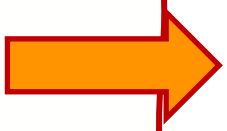
local variable

```
g++ -o main main.cpp
main.cpp: In function 'int other_function()':
main.cpp:12:3: error: 'loc' was not declared in this scope
   loc = 2;  // wrong: loc is not visible from this function
   ^~~
main.cpp:12:3: note: suggested alternative: 'glob'
   loc = 2;  // wrong: loc is not visible from this function
   ^~~
   glob
main.cpp: In function 'int main()':
main.cpp:18:3: error: 'loc' was not declared in this scope
   loc = 2;  // wrong: loc is not visible from this function
   ^~~
```

Intermezzo (1) : variables scope

https://cplusplus.com/

```
int glob;         // global variable

int some_function ()
{
  int loc;      // local variable
  loc = 0;
}

int other_function ()
{
  glob = 1;  // ok: glob is a global variable
  loc  = 2;  // wrong: loc is not visible from this function
}

int main ()
{
  glob = 1;  // ok: glob is a global variable
  loc  = 2;  // wrong: loc is not visible in the main
}
```



```
int glob;         // global variable

int some_function ()
{
  int loc;      // local variable
  loc = 0;
}

int other_function ()
{
  glob = 1;     // ok: glob is a global variable
  int loc = 2;  // ok: loc is a different local variable !
}

int main ()
{
  glob = 1;     // ok: glob is a global variable
  loc  = 2;     // wrong: loc is not visible in the main
}
```

Intermezzo (1) : variables scope

https://cplusplus.com/

```cpp
#include <iostream>
int glob = 2 ;

int main () {

  std::cout << "The value of glob is " << glob << std::endl;

  for ( int k = 0 ; k < 10 ; k++ ) {
    double x = double(k) + 3 ;
    std::cout << "The value of x is " << x << std::endl;
  }

  std::cout << "The value of x is " << x << std::endl; // WRONG !

}
```

```
> g++ -o main main.cpp
main.cpp: In function ‘int main()’:
main.cpp:33:39: error: ‘x’ was not declared in this scope
  std::cout << "The value of x is " << x << std::endl;
                                       ^
```

x is a “local variable” that leaves inside the for loop only !

Intermezzo (1) : variables scope, try to check with this ! https://cplusplus.com/

Global variable : no scope, it can be accessed and modified everywhere

This variable is defined in the main, can be accessed from everywhere inside the main

This variable is defined in the for(){ }, can be accessed only there (doesn't exist outside)

```
#include <iostream>

using namespace std;

double var_global = 3 ;

void myFunction( double var_input ) {

  double var_infun = 2 ;
  cout << " Global = " << var_global
       << " input = " << var_input
       << " var_in_function  = " << var_infun
       << " local1 = " << local1 << endl;
       << endl;

}

int main() {

  double var_local1 = 4 ;

  for ( int k = 0 ; k < 10 ; k++ ) {
    double var_local2 = k ;
    cout << "Global = " << var_global
         << " local1 = " << var_local1
         << " local2 = " << var_local2 << endl;
  }

  cout << "Global = " << var_global
       << " local1 = " << var_local1
       << " local2 = " << var_local2
       << " var_in_function = " << var_infun << endl;

  myFunction( var_local1  ) ;

}
```

Each variable lives inside the scope where it has been declared

These would cause a compilation error

6. Reading from and writing to files

```
#include <iostream>
#include <fstream>

using namespace std;

int main() {

  ifstream inputFile("input.txt") ;
  ofstream outputFile("output.txt") ;

  if(!inputFile){
    cout <<"Error opening input file: exiting" <<endl;
    return -1;
  }

  double a ;

  inputFile >>a;
  outputFile << a;

  inputFile.close();
  outputFile.close();

  return 0;

}
```

`<fstream>` is needed to communicate with files

Create an `ifstream` variable called "inputFile" (constructor which accepts a file name as input) : this is used to read from file

Create an `ofstream` variable called "outputFile" (constructor which accept sa file name as input) : this is used to write to file

Check is the file is opened (good to use this check to avoid problems in the program execution)

Read from and write to files

Closes the file currently associated with the object, disassociating it from the stream (not strictly necessary here as the destructor would do the same)

6. Reading from and writing to files

```cpp
#include <iostream>
#include <fstream>

using namespace std;

int main() {

  ifstream inputFile("input.txt") ;
  ofstream outputFile("output.txt") ;

  if(!inputFile){
    cout <<"Error opening input file: exiting" <<endl;
    return -1;
  }

  double a ;

  inputFile >>a;
  outputFile << a;

  inputFile.close();
  outputFile.close();

  return 0;

}
```

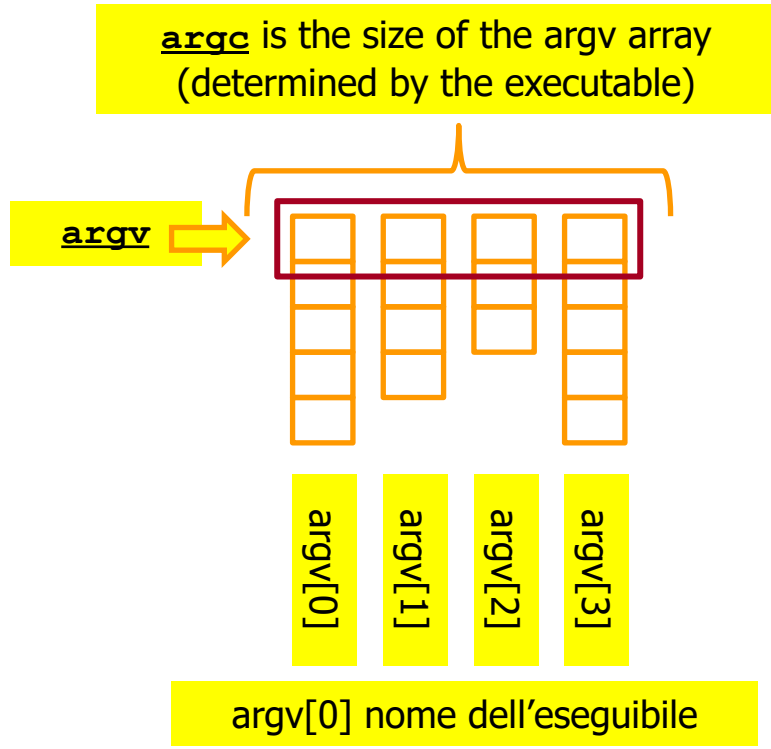
This program reads a number from a fixed file called `input.txt`. Suppose I want to read numbers from different files, what should I do ?

- Open the code
- Modify the name of the input file (`input1.txt` or `input2.txt` ..)
- Recompile
- Run

Is there something a bit smarter I can do ? Try to parametrise the input filename (see next slides)

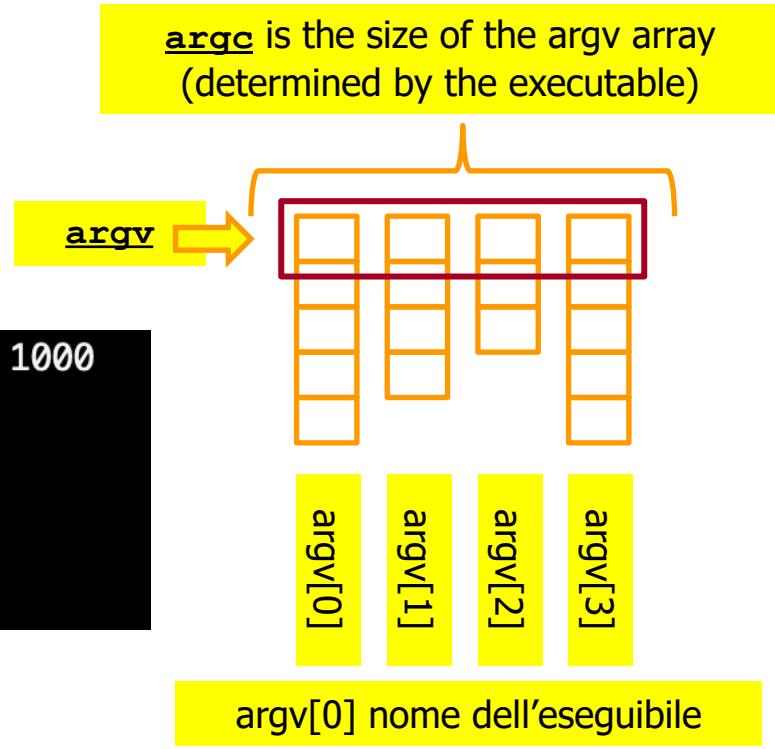
7. Program communication with outside : `argv (argc)`

```cpp
#include <iostream>

using namespace std;

int main( int argc , char** argv ) {

  cout << argc << endl;
  for ( int k = 0 ; k < argc ; k++ ) cout << argv[k] << endl;

  return 0 ;

}
```



7. Program communication with outside : `argv (argc)`

```
Leonardos-MBP-4:Lezione1 lcarmina$ ./hw_argc_1 data.txt 1000
3
./hw_argc_1
data.txt
1000
Leonardos-MBP-4:Lezione1 lcarmina$
```



7. Program communication with outside : `argv (argc)`

```cpp
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

int main( int argc , char** argv ) {

  if ( argc < 3 ) {
    cout << "Uso del programma : " << argv[0] << " <ndati> <input_filename> <output_filename>" << endl;
    return -1;
  }

  cout << argc << endl;
  for ( int k = 0 ; k < argc ; k++ ) cout << argv[k] << endl;

  ifstream inputFile( argv[2] ) ;
  ofstream outputFile( argv[3]) ;
  unsigned int ndati = atoi( argv[1] );

  if(!inputFile){
    cout <<"Error opening input file: exiting" <<endl;
    return -1;
  } else {
    for ( int k = 0 ; k < ndati ; k++ ) {
      double dato ;
      inputFile >> dato;
      if ( inputFile.eof() ) {
        cout << "End of file reached, exiting" << endl;
        return -1;
      }
      cout << "Reading number " << dato <<endl;
      outputFile << dato << endl;
    }
  }

  inputFile.close();
  outputFile.close();

  return 0 ;

}
```

Accept inputs from outside

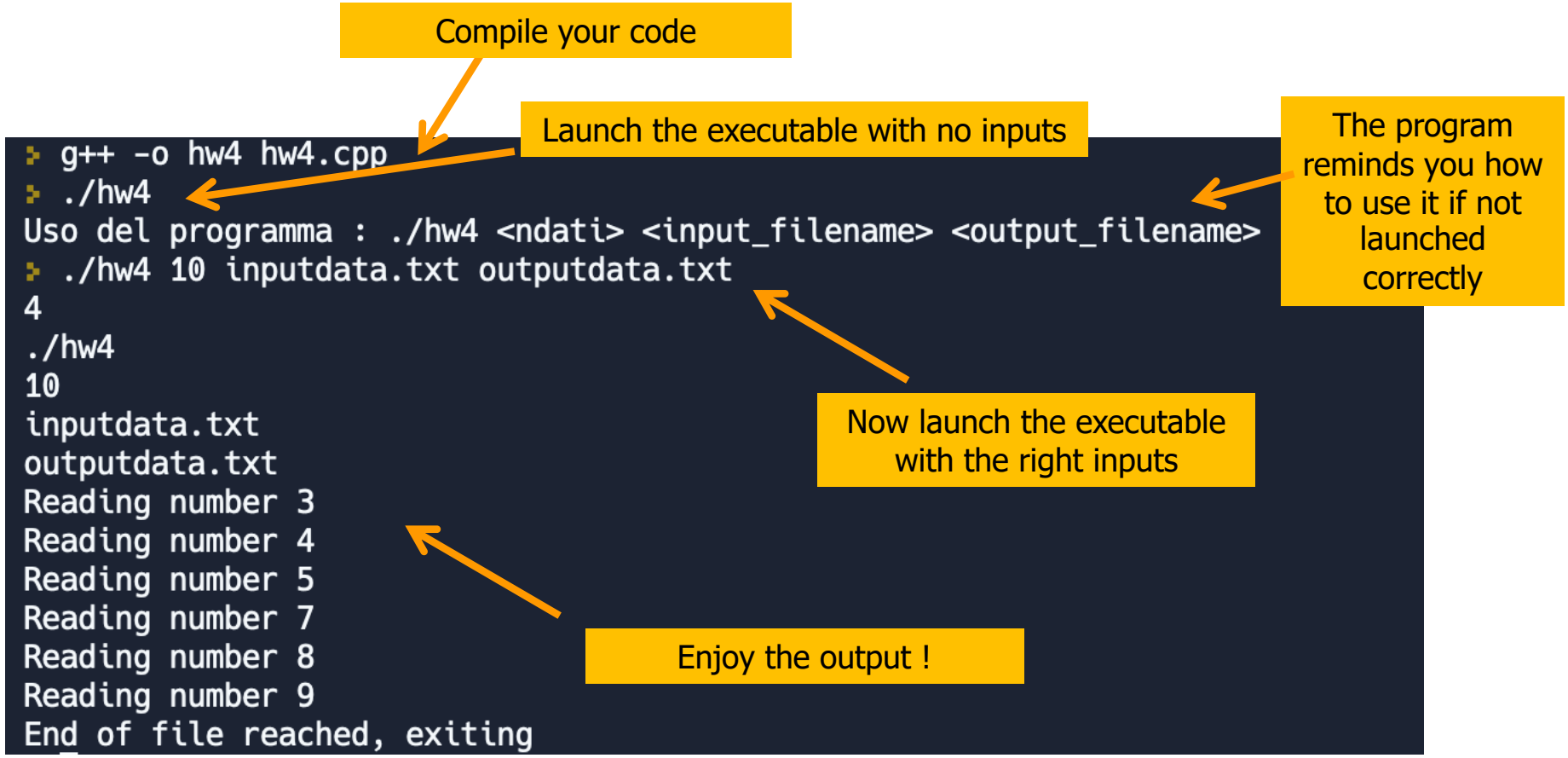
Check that all required arguments are there

Number of elements to read is passed as a `char*`, need to convert it into integer (use `atoi` from `cstdlib`)

Another useful check is if by chance the end of input file is reached (can't read more elements than the ones written in the file)

`return 0` means 'success', return any other number for 'failure'

7. Program communication with outside : how it works



8. The most naïve container

```
#include <iostream>
#include <cstdlib>

using namespace std;

int main( int argc , char** argv) {

  if ( argc < 2 ) {
    cout << "Uso del programma : " << argv[0] << " <nelements>" << endl;
    return -1;
  }

  int nelements = atoi(argv[1]);

  double vfixed[nelements] ;

  for ( int k = 0 ; k < nelements ; k++ ) {
    vfixed[k] = k*k ;
  }

  double * vdynamic = new double[nelements] ;

  for ( int k = 0 ; k < nelements ; k++ ) {
    vdynamic[k] = k*k ;
  }

  for ( int k = 0 ; k < nelements ; k++ ) {
    cout << "Component k = " << vdynamic[k] << endl;
  }

  delete [] vdynamic ;

  return 0 ;

}
```

`vfixed` has a size which is set at compilation time : set to what ? To the content of `nelements` which is undefined. This will create an array of undefined size !

`vdynamic` has a size which is set at runtime : to your exact desired value `nelements` which is correctly filled at execution time

don't forget to release the allocated memory (at this point is not that important because we are close to the end of the program)

Memory leak !

Suppose you need to loop over 1M files, read a bunch of numbers from each files

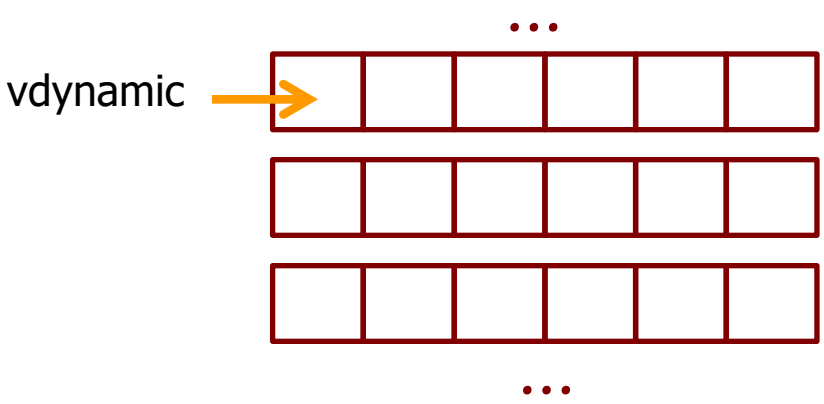
```
for ( int nfiles = 0 ; nfiles < 1000000 ; nfiles++ ) {

  ifstream fin( filenames[k] );

  double * vdynamic = new double[nelements] ;
  double dato;

  for ( int k = 0 ; k < nelements ; k++ ) {
    fin >> dato;
    vdynamic[k] = data;
  } ;

  delete[] vdynamic ;

}
```

Loop on the list of files

Allocate dynamically a data container

Fill the container

Don't forget to clean up memory !



- `vdynamic` is a pointer to the first element of the array
- When the pointer goes out of scope the pointer is cancelled but the allocated memory is not ! Memory allocated for the array is declared as filled and there's no way to access it anymore
- So, each new pointer will keep allocating new memory and leave it occupied
- Memory leak : the program will crash when the physical memory of the machine is completely filled
- Use smart pointers !

esercizio1.0.cpp : quick and dirty

```cpp
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

int main ( int argc, char** argv ) {

  if ( argc < 3 ) {
    cout << "Uso del programma : " << argv[0] << " <n_data> <filename> " << endl;
    return -1 ;
  }

  // open the file "1941.txt" and read elements into a dynamic array

  // ...

  // compute the mean and standard deviation

  // ...

  // order the array and compute the median

  // ...

  // write the ordered array into a file

  // ...

  return 0

}
```

esercizio1.0.cpp : quick and dirty

```cpp
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

int main ( int argc, char** argv ) {

  if ( argc < 3 ) {
    cout << "Uso del programma : " << argv[0] << " <n_data> <filename> " << endl;
    return -1 ;
  }

  // open the file "1941.txt" and read elements into a dynamic array

  // ...

  // compute the mean and standard deviation

  // ...

  // order the array and compute the median

  // ...

  // write the ordered array into a file

  // ...
```

then compile in this way

```
g++ esercizio1.0.cpp -o esercizio1.0
```

Compilation can be broken into three main parts

1. pre-processing
2. actual compilation
3. linking

It is the phase in which the directives to the compiler are managed:

- #Include expansion: the compiler loads the declarations of the functions defined in the included libraries in order to check that these functions are used correctly in the program
- Replacing constants defined with #define: the compiler searches for all occurrences of these constants in the program and replaces them with the corresponding values Management of other directives that we will not see ...

Try `gcc -E esercizio1.0.cpp`

Compilation can be broken into three main parts

1. pre-processing
2. actual compilation
3. linking

It is the phase in which the program is transformed into binary code (not yet in an executable). It consists of several sub-phases (not necessarily sequential):

- Type control check that the variables used have been declared, check that operations in expressions are used with arguments of an appropriate type, check that the functions are called with number and type parameters appropriate…
- Analysis and optimizations : elimination of dead code, optimisation of the code…
- Generation of the binary code

Try `g++ -c esercizio1.0.cpp`

Compilation can be broken into three main parts

1. pre-processing
2. actual compilation
3. linking

Multiple object modules are linked together to create an executable file

- file1.o file2.o ...: different modules object of the same program obtained through separate compilation (we will see shortly)
- standard and system libraries
- libfile1.a libfile2.a ...: several external libraries that provide functions used within the program The result of this phase is a single executable file

Try `g++ -o esercizio esercizio1.0.cpp`

esercizio1.1.cpp (split into functions)

```cpp
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

double CalcolaMedia( double * , int ) ;
double * ReadDataFromFile ( const char*  , int ) ;
void Print ( const char* , double * , int ) ;

int main ( int argc , char** argv) {

  if ( argc < 2 ) {
    cout << "Uso del programma : " << argv[0] << " <n_data> <filename> " << endl;
    return -1 ;
  }

  int ndata = atoi(argv[1]);
  char * filename = argv[2];

  double * data = ReadDataFromFile ( filename, ndata ) ;
  cout << "Media = " << CalcolaMedia( data , ndata ) << endl;
  Print( "fileout.txt", data, ndata ) ;

}

double * ReadDataFromFile ( const char* Filename , int size  ) {
  double * data = new double[size];
  //   [ ... ]
  return data;
}

void Print ( const char* Filename, double * data, int size ) {
  //  [ ... ]
}

double CalcolaMedia( double * data , int size ) {
  //   [ ... ]
  return media ;
}
```

Functions declarations: input and return type only are important at this stage

The main simply calls the functions

Functions implementation : code how the function should work

esercizio1.1.cpp (split into functions)

```cpp
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

double CalcolaMedia( double * , int ) ;
double * ReadDataFromFile ( const char*  , int ) ;
void Print ( const char* , double * , int ) ;

int main ( int argc , char** argv) {

  if ( argc < 2 ) {
    cout << "Uso del programma : " << argv[0] << " <n_data> <filename> " << endl;
    return -1 ;
  }

  int n
  char *

  double
  cout <
  Print(

}

double * ReadDataFromFile ( const char* Filename , int size  ) {
  double * data = new double[size];
  //   [ ... ]
  return data;
}

void Print ( const char* Filename, double * data, int size ) {
  //  [ ... ]
}

double CalcolaMedia( double * data , int size ) {
  //   [ ... ]
  return media ;
}
```

then compile in this way

```
g++ esercizio1.1.cpp -o esercizio1.1
```

esercizio1.2.cpp : move functions outside

```cpp
#include <iostream>
#include <fstream>
#include <cstdlib>

using namespace std;

double CalcolaMedia( double * , int ) ;
double * ReadDataFromFile ( const char*  , int ) ;
void Print ( const char* , double * , int ) ;

int main ( int argc , char** argv) {

  if ( argc < 2 ) {
    cout << "Uso del programma : " << argv[0] << " <n_data> <filename> " << endl;
    return -1 ;
  }

  int ndata = atoi(argv[1]);
  char * filename = argv[2];

  double * data = ReadDataFromFile ( filename, ndata ) ;
  cout << "Media = " << CalcolaMedia( data , ndata ) << endl;
  Print( "fileout.txt", data, ndata ) ;

}

double * ReadDataFromFile ( const char* Filename , int size  ) {
  double * data = new double[size];
  //   [ ... ]
  return data;
}

void Print ( const char* Filename, double * data, int size ) {
  //  [ ... ]
}

double CalcolaMedia( double * data , int size ) {
  //   [ ... ]
  return media ;
}
```

Move the functions declarations into a dedicated file (header file, funzioni.h) and replace this with #include "funzioni.h"

The main doesn't change

Move the functions implementation into a dedicated file (funzioni.cpp or funzioni.cxx)

esercizio1.2.cpp : move functions outside

```cpp
#include <iostream>
#include <fstream>
#include <cstdlib>

#include "funzioni.h"

using namespace std;

int main ( int argc , char** argv) {

  if ( argc < 2 ) {
    cout << "Uso del programma : " << argv[0] << " <n_data> <filename> " << endl;
    return -1 ;
  }

  int ndata = atoi(argv[1]);
  char * filename = argv[2];

  double * data = ReadDataFromFile ( filename, ndata ) ;
  cout << "Media = " << CalcolaMedia( data , ndata ) << endl;
  Print( "fileout.txt", data, ndata ) ;

}
```

Move the functions declarations into a dedicated file (header file, funzioni.h) and replace this with #include "funzioni.h"

then compile in this way

```
g++ -o esercizio1.2 esercizio1.2.cpp funzioni.cpp
```

Makefile

The makefile helps you in organizing the compilation instructions: just create a new file called Makefile

```
esercizio1.2 : esercizio1.2.cpp funzioni.cpp funzioni.h
	g++ esercizio1.2.cpp funzioni.cpp -o esercizio1.2

clean:
	rm esercizio1.2
```

- Structure of the makefile

<target> : <dep1> <dep2> <dep3> …
[tab] <compilation instruction>

```
esercizio1.2 : esercizio1.2.o funzioni.o
	g++ esercizio1.2.o funzioni.o -o esercizio1.2

funzioni.o: funzioni.cpp funzioni.h
	g++ -c funzioni.cpp -o funzioni.o

esercizio1.2.o : esercizio1.2.cpp funzioni.h
	g++ -c esercizio1.2.cpp -o esercizio1.2.o

clean:
	rm esercizio1.2
	rm *.o
```

- The name of the target must match the name of the output file
- Dependencies : list of files that needs to be checked to trigger a new compilation. The timestamps of the dependencies are compared to the timestamp of the output file.
- The option –c forces g++ to compile but not link
- To launch a compilation just type "make esercizio1.2" (if you simply write make it will execute the first target)

Only the parts of the programs which have been modified are recompiled !

Makefile

Some useful parameters of the gcc compiler are as follows:

- -o: Allows you to specify the name of the executable file to be generated (as an alternative to a.out.
  - Usage: g++ -o esercizio1.0 esercizio1.0.cpp (g++ esercizio1.0.cpp -o esercizio1.0)
- -Wall: Enables the display of all warning messages generated during compilation
- -pedantic: Check that the program exactly meets the rules of the C standard (ISO C). Report any violations of the standard with warning messages
- -c : compilation but no linking

It is always good to use gcc at least as follows (might need to add –c if needed):

g++ -Wall -o esercizio1.1 esercizio1.1.cpp funzioni.cpp

What's in today menu

Take away messages from today lecture (in random order):

- What a namespace is
- Variables scope
- How to pass inputs to your executable (argv,argc)
- Fixed size array (at compilation level) vs variable size array (at run-time)
- Break the code into functions, how variables enter (reference, value and pointer) and how they exit. Break into different files accordingly
- Read/write to files
- Compilation and makefiles

Intermezzo (2) : what exactly std:: means ? Namespaces

Two variables with the same name in the same scope cannot exist. In each scope, a name can only represent one entity. For example, there cannot be two variables with the same name in the same scope

```
int some_function ()
{
  int x;
  x = 0;
  double x;   // wrong: name already used in this scope
  x = 0.0;
}
```

- This is seldom a problem for local names, since blocks tend to be relatively short, and names have purposes within them, such as naming a counter variable, an argument, etc...

- But non-local names bring more possibilities for name collision, especially considering that libraries may declare many functions, types, and variables, neither of them local in nature, and some of them very generic.

- Namespaces allow us to group named entities that otherwise would have *global scope* into narrower scopes, giving them *namespace scope*. This allows organizing the elements of programs into different logical scopes referred to by names.

```
namespace myNamespace
{
  int a, b;
}
```

Intermezzo (2) : what exactly std:: means ? Namespaces

Using namespaces, we can create two variables or member functions having the same name. Namespaces allow us to group named entities that otherwise would have *global scope* into narrower scopes, giving them *namespace scope*.

```
#include <iostream>

namespace rightmath {
  const int one = 1;
  const int two = 2;
  int add ( const int x, const int y ) { return x+y } ;
};

namespace wrongmath {
  const int one = 2;
  const int two = 1;
  int add ( const int x, const int y ) { return x-y } ;
};

using namespace rightmath;
using namespace std;

int main() {
  std::cout << "Addizione corretta :"
            << rightmath::one << " + " << rightmath::two << " = " <<
            << rightmath::add( rightmath::one , wrongmath::two ) << std::endl;
  cout << "Addizione sbagliata :"
       << wrongmath::one << " + " << wrongmath::two << " = " <<
       << add( wrongmath::one , wrongmath::two ) << std::endl;
}
```

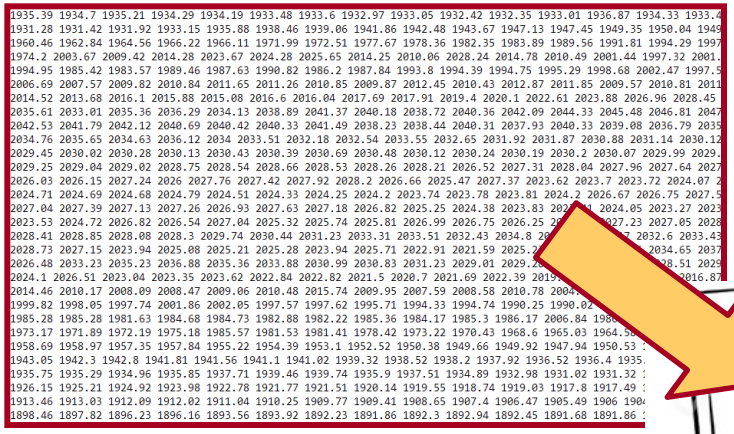
The effect of the keyword `using` : if not specified use "rightmath" scope

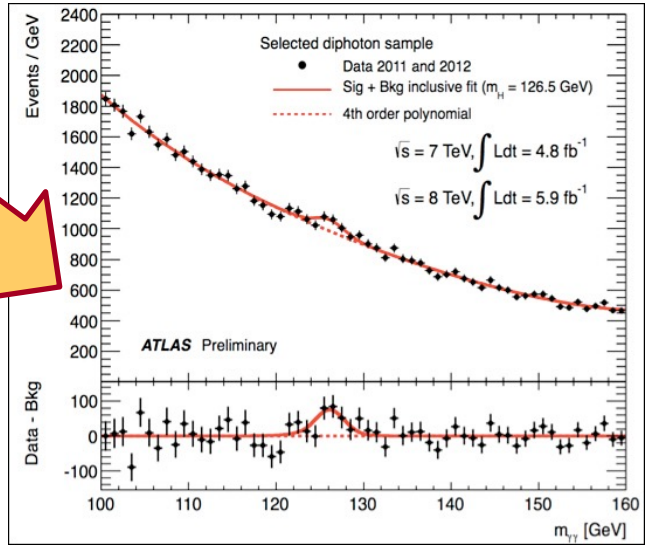
More variants

```
if(!inputFile){
  cout <<"Error opening input file: exiting" <<endl;
  return -1;
} else {
  for ( int k = 0 ; k < ndati ; k++ ) {
    double dato ;
    inputFile >> dato;
    if (inputFile.eof() ) {
      cout << "End of file reached, exiting" << endl;
      return -1;
    }
    cout << "Reading from input file :" << dato << endl;
    outputFile << dato << endl;
  }
}
```

```
int nelements = 0;
if(!inputFile){
  cout <<"Error opening input file: exiting" <<endl;
  return -1;
} else {
  while (true) {
    double dato;
    inputFile >> dato;
    if( inputFile.eof() ) break;
    nelements++;
    cout << "Reading from input file :" << dato << endl;
    outputFile << dato << endl;
  };
} ;
```

The problem



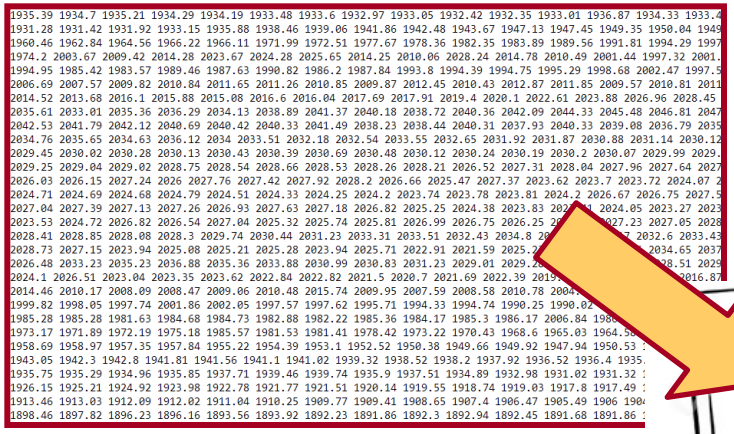


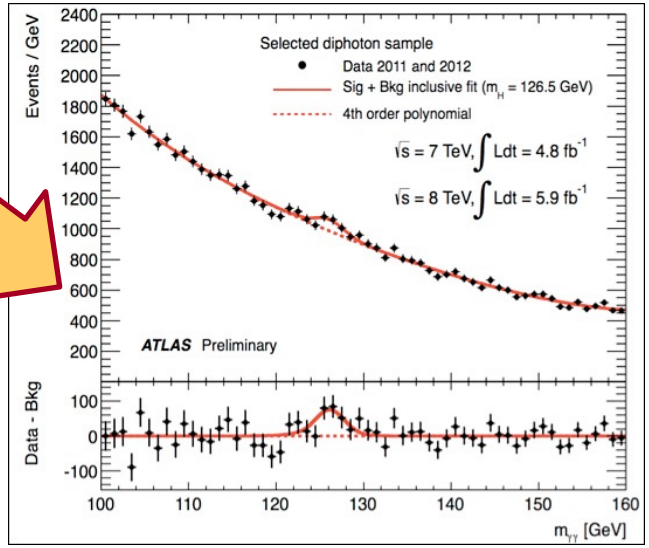
Our goal in the first 3 lessons will be to analyze a dataset (a bunch of numbers)

- ❑ Data container
- ❑ Operations on data

Average, variance, median

The problem





Lesson 1 : read a **given number** of elements from a file. The number of elements to be read **is passed by the user at runtime** (not compilation time !)

Average, variance, median