Esercitazioni
Laboratorio II - I modulo (calcolo)

Contenuto del corso

- Introduzione alla programmazione in C++
- Funzioni di base del linguaggio C++ e primi programmi
- Implementazione di semplici algoritmi in C++
- La programmazione a oggetti
- L'uso dei template
- Ereditarietà e polimorfismo
- Introduzione a ROOT
- Esercizi al calcolatore relativi alla teoria statistica

Documentazione

- http://lab-info.blogspot.com/
  - per scambiare informazioni, suggerimenti, soluzioni
- http://lab-info-q.blogspot.com/
  - per dettagli ed approfondimenti
- Per contattarci :
  - maura.pavan@mib.infn.it
  - luigi.zanotti@mib.infn.it
  - andrea.biancini@mib.infn.it
  - davide.chiesa@mib.infn.it

Introduzione al C++

Sorgente ed eseguibile

E' necessario avere ben presente la distinzione tra:

- **Codice Sorgente**: versione “leggibile dagli esseri umani”
- **Codice Eseguibile**: versione che il pc può leggere ed eseguire

Il compito di trasformare la prima nella seconda è assolto dal **compilatore**, nel nostro caso c++ (o g++)

- Come scriviamo il Codice Sorgente?

  Editor di testo → (Gedit, Emacs, Kate, Vim...)

- Come lo compiliamo?

  c++ -o <nome eseguibile> <nome_sorgente>

Primo programma in C++

- Scriviamo il seguente programma:

```
/*  Programma #1 - Un primo progromma in C++.
    Scrivete questo programma in un file,
    compilatelo e avviatelo
*/
#include <iostream>

int main()
{
  std::cout << "This is my first C++ program.";
  return 0;
}
```

- Compiliamo ed avviamo da terminale, così:

```
marco@barattolo:2011$ c++ -o esempio_1_00 esempio_1_00.cpp
marco@barattolo:2011$ ./esempio_1_00
This is my first C++ program.marco@barattolo:2011$
```

Analizziamo il codice sorgente

Commenti ignorati dal compilatore, molto utili per il programmatore. Altro metodo, usare //, ma vale solo per una riga!

Inclusione di header (libreria): molte funzioni di cui avremo bisogno sono contenute in librerie messe a disposizione dal compilatore

```
/*  Programma #1 - Un primo progromma in C++.
    Scrivete questo programma in un file,
    compilatelo e avviatelo
*/
#include <iostream>

int main()
{
  std::cout << "This is my first C++ program.";
  return 0;
}
```

Dichiarazione della funzione principale, obbligatoria. Ogni programma inizia chiamando *main*. Preceduto da dichiarazione del tipo di valore ritornato (*int* in questo caso)

- std::cout → chiamata a funzione *cout* (console output) contenuta nelle librerie standard *std*
- << → operatore di output
- "This... " → stringa di testo
- ; → tutte le istruzioni terminate dal punto e virgola

Termina il programma e ritorna il valore 0 (intero) al processo che ha invocato il programma stesso (tipicamente il sistema operativo)

Un programma più completo...

- Provate il seguente programma:

```
#include<iostream>

int main (int numArg, char *listArg[])
{
  std::cout << "nome del programma: "
            << listArg[0] << std::endl ;

  int num;
  std::cout << "inserisci un numero "
            << std::endl ;
  std::cin  >> num ;
  std::cout << "il numero inserito è: "
            << num << std::endl ;

  return 0 ;
}
```

- Unici argomenti possibili per la funzione *main* sono:
  - numero di argomenti passati da riga di comando (int)
  - lista degli argomenti (char)
- Operatore di input >>
- Funzione *endl*

- Qual è il primo argomento passato al *main* dal SO?
- Provate a passare altri argomenti al programma e farvi restituire il numero totale di argomenti

Le variabili

- Quantità di interesse si gestiscono come variabili

```
int num1 = 0 ;
int num2 = 3 ;
int somma = num1 + num2 ;
std::cout << "somma: "
          << somma << "\n" ;
float razionale = 3.1416 ;
double razionale2 = 1.4142 ;
char lettera = 'a' ;
bool condizione = true ;
```

```
int vettore[10] ;
for (int i = 0 ; i < 10 ; ++i)
  {
    vettore[i] = i * 2 ;
  }
```

- Diversi oggetti sono gestiti da diversi **tipi** (int, float, char...)
- Una variabile è **l'istanza di un tipo** (num1 è una istanza di int)
- Le normali operazioni che ci si aspetta sono definite sui tipi
- Si possono creare vettori di un tipo di variabili, chiamati **array**
- La lista degli oggetti di un array occupa una zona contigua della memoria
- La dimensione di un array è una costante

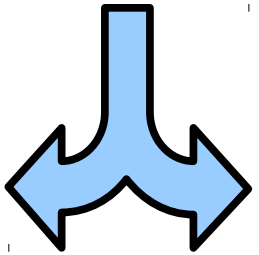
Le variabili

Type	Bit Width	Common Range
char	8	–128 to 127
unsigned char	8	0 to 255
signed char	8	–128 to 127
int	16	–32,768 to 32,767
unsigned int	16	0 to 65,535
signed int	16	–32,768 to 32,767
short int	16	same as int
unsigned short int	16	same as unsigned int
signed short int	16	same as short int
long int	32	–2,147,483,648 to 2,147,483,647
unsigned long int	32	0 to 4,294,967,295
signed long int	32	–2,147,483,648 to 2,147,483,647
float	32	3.4E–38 to 3.4E+38
double	64	1.7E–308 to 1.7E+308
long double	80	3.4E–4932 to 1.1E+4932
bool	N/A	true or false
wchar_t	16	0 to 65,535

Gli operatori

- Avete a disposizione diversi operatori definiti in c++:
  - operatori **aritmetici**: +, -, *, /, %
  - operatori di **incremento e decremento**: ++, --
  - operatori **logici e relazionali**: >, >=, <, <=, ==, !=, &&, ||, !
- Fate sempre attenzione alla **precedenza** tra operatori...

Più alta	**++ --**
	/ * %
Più bassa	**+ -**



Più alta	**!**
	> >= < <=
	== !=
	&&
Più bassa	**||**

- ...e alla **conversione** tra tipi delle espressioni aritmetiche → in una operazione tra tipi diversi, tutti gli operandi vengono convertiti al tipo "più grande" **prima** di compiere l'operazione

Suggerimenti e esercizi

- Fate uso delle parentesi nelle espressioni matematiche!
- Potete utilizzare, se necessario, un ***cast*** tra tipi. La sintassi generica è *(tipo) espressione*
  - Ad esempio *(float) x / 2* vi restituirà un tipo float come risultato

- **Esercizio 1**: scrivete un programma che, letti due numeri interi da terminale, restituisca il loro rapporto
- **Esercizio 2**: scrivete un programma che, lette due variabili booleane da terminale, restituisca il risultato delle operazioni AND, OR e XOR
  (suggerimento: il tipo *bool* in c++ accetta solo due valori, *true* e *false*, ma questi sono del tutto equivalenti a 1 ed 0)

Strutture di controllo: for

- Un'operazione da ripetere molte volte può essere automatizzata:

```
for (int index = 0 ; index < numArg ; ++index)
  {
    std::cout << index << "-esimo argomento: "
              << listArg[index]
              << std::endl ;
  }
```

- Le istruzioni nello scope vengono eseguite finchè **index < numArg**
- La variabile **index** esiste soltanto nello scope del ciclo
- Il ciclo viene impostato **definendo** index, stabilendo la **condizione di uscita** e dettando l'istruzione di **incremento** della variabile di controllo

Strutture di controllo: while

- Il **for** non è l'unico modo per realizzare un ciclo:

```
int index = 0 ;
while (index < numArg)
  {
    std::cout << index << "-esimo argomento: "
              << listArg[index]
              << std::endl ;
    ++index ;
  }
```

- Le istruzioni nello scope vengono eseguite finché **index < numArg**
- La variabile **index** esiste anche al di fuori del ciclo
- E' necessario definire correttamente la variabile di controllo prima dell'inizio del ciclo

Strutture di controllo: if

- Eseguire istruzioni solo se verificata una condizione

```
if (numArg > 1)
  {
    std::cout << "primo argomento opzionale: "
              << listArg[1] << std::endl ;
  } else {
    std::cout << "non ci sono argomenti opzionali"
              << std::endl ;
  }
```

- Se l'espressione di controllo è vera, viene eseguita la sequenza di istruzioni nel primo *scope* (le parentesi {})
- Altrimenti viene eseguita la sequenza dopo ***else***
- La presenza di *else* non è obbligatoria!

Strutture di controllo: switch

- Se le possibilità non sono due (vero/falso), ma **molte**

```
switch (x)
{
  case 1:
    std::cout << "x vale 1";
    break;

  case 2:
    std::cout << "x vale 2";
    break;

  default:
    std::cout << "x non vale nè 1 nè 2";
}
```

- La variabile **x** è già definita
- Per ogni possibile valore c'è un'istruzione da eseguire
- L'istruzione **break** è necessaria per terminare subito lo **switch**
- Il **default** è l'istruzione che viene eseguita se nessuno dei casi precedenti è vero

Operazioni matematiche

- Operatori matematici non presenti nelle librerie standard sono forniti dalle librerie matematiche

```
#include<iostream>
#include<cmath>

int main (int numArg, char *listArg[])
{
  double due = 2 ;
  double radice_due = sqrt (due) ;
  std::cout << radice_due << std::endl ;
}
```

- E' necessario includere la libreria **cmath** per avere a disposizione le funzioni necessarie
- Le funzioni matematiche sono chiamate nel programma

Esercizi

- **Esercizio 3**: Scrivere un programma che scrive a terminale
  - la radice quadrata di 2, il cubo di 2 il seno di pigreco/4
- **Esercizio 4**: Scrivere un programma che:
  - controlli se uno o più argomenti opzionali sono passati al main al momento dell'avvio (e in caso contrario, avvisi l'utente)
  - scriva l'intera lista di argomenti opzionali passati al main
  - chieda all'utente di inserire un intero a scelta tra 1 e 2, e restituisca a terminale il valore inserito, o un messaggio di errore in caso di inserimento di altri interi
- **Esercizio 5**: Scrivere un programma che, dato un array di n interi casuali, lo ordini dal più piccolo al più grande (suggerimento: per generare i numeri casuali, usate la funzione *rand()* contenuta in *cstdlib*)

Ordine!

- **SINTASSI** = insieme di sane abitudini, oltre alla scrittura corretta del codice. Ad esempio:
  - **indentare** il codice
  - **commentare** il codice (per se stessi e per gli altri)
  - scegliere un **modo coerente** di scrivere il codice (parentesi graffe a capo o non a capo, spaziature...)
  - scegliere **nomi lunghi** ed **esplicativi** per le variabili