Appunti Lezione 02

Lorenzo Visca

1 Passaggio by reference

Codice di riferimento: `ioexample2.C`

In `ioexample.C` la macro prende in input i nomi dei file *by value*. In questo modo la macro crea una copia delle stringhe ricevute, ed eventuali modifiche non si riflettono sul file chiamante. La sintassi è:

```
void ioexample(string fimpName, string foupName)
```

In alternativa si può passare la stringa *by reference*, in questo modo non vengono create copie ma si lavora su un puntatore alla stringa originale, eventuali modifiche quindi si riflettono sul valore assunto nel programma chiamante.
Se l'oggetto passato non viene modificato è buona norma dichiararlo con il prefisso `const`, in questo modo si evita che venga modificato accidentalmente. La sintassi è:

```
void ioexample2(const string& fimpName, const string& histName)
```

2 Istogrammi, dichiarazione dinamica

Codice di riferimento: `ioexample2.C`

La classe `TH1D` permette di creare istogrammi di valori `double`. La sintassi utilizzata per creare un istogramma è la seguente:

```
TH1D* hist = new TH1D("hist", "Istogramma", 100, -7., 7.);
```

Dove i parametri utilizzati sono:

- `"hist"`: nome dell'istogramma, è il nome con cui ROOT gestisce l'istogramma in memoria. Solitamente si usa lo stesso nome dato alla variabile (come in questo caso), ma non è obbligatorio;
- `"Istogramma"`: titolo dell'istogramma;
- `100`: numero di bin;
- `-7.,7.`: valore minimo e valore massimo dell'asse x.

La sintassi `TH1D* hist = new TH1D(...)` dichiara l'istogramma in modo dinamico: `hist` è un puntatore ad un oggetto situato nella *heap memory*, più grande della *stack memory*, più piccola e potenzialmente scomoda per oggetti di grandi dimensioni.
La principale differenza è che gli oggetti nella heap memory non vengono distrutti automaticamente quando si esce dallo scope (ovvero quando si raggiunge la parentesi graffa di chiusura), ma devono essere distrutti manualmente con il comando `delete` (non facendolo si rischia di creare *memory leaks*):

```
double* x = new double(3.14);
delete x;
```

Questa proprietà della heap memory è fondamentale per visualizzare l'istogramma, che in questo modo non viene cancellato quando termina l'esecuzione della macro. In realtà il comando `delete` non deve essere utilizzato per oggetti ROOT (`TObject`), in quanto ROOT gestisce automaticamente la memoria degli oggetti creati. Se si utilizzasse `delete` su un `TObject` poi ROOT proverebbe a sua volta a cancellarlo, generando un errore.
A titolo di esempio, la dichiarazione dello stesso istogramma nella stack memory sarebbe:

```
TH1D hist("hist", "Istogramma", 100, -7., 7.);
```

A livello di sintassi c'è una differenza nell'utilizzo delle member functions delle varie classi ROOT:

- se l'oggetto è un puntatore (come nel caso di `TH1D* hist`), per accedere alle member functions si usa l'operatore `->`

  ```
  TH1D* hist = new TH1D(...);
  hist->Draw();
  ```

- mentre se l'oggetto non è un puntatore (come nel caso di `TH1D hist`) si usa l'operatore `.`

  ```
  TH1D hist(...);
  hist.Draw();
  ```

3 Creazione di un file ROOT

Codice di riferimento: `ioexample2.C`

Le righe dalla 31 alla 34 creano un file ROOT e ci salvano all'interno l'istogramma con il comando `Write();` il file è un oggetto della classe`TFile`, in questo caso la dichiarazione è statica e quindi la chiusra del file avviene con la sintassi:

```
file.Close();
```

Gli argomenti passati al costruttore della classe `TFile` sono:

- il nome del file come array di char (la trasformazione da `string` ad array di char avviene tramite la funzione `c_str()`), l'estensione utilizzata per i file ROOT è `.root`;
- la modalità di apertura del file: in questo caso `"recreate"` indica che se il file esiste già viene sovrascritto, altrimenti viene creato.

La funzione `Write()` salva automaticamente l'istogramma all'interno della current working directory, che in questo caso è il file appena creato.
La macro creata viene caricata ed eseguita tramite:

```
root[ ]  .L ioexample2.C
root[ ]  ioexample2("dati0.dat","output.root")
```

4 Gestione alternativa dei file ROOT

Codice di riferimento: `ioexbis.C`

In `ioexample2.C` il `TFile` viene creato dopo l'istogramma, e il salvataggio avviene tramite la member function `Write()` dell'istogramma.
Questa tecnica diventa scomoda se si vogliono salvare numerosi oggetti nel file, in quanto bisognerebbe chiamare `Write()` per ogni oggetto. Un'alternativa è dichiarare il file prima dell'istogramma, in questo modo ogni oggetto creato successivamente si trova nella working directory del file, è quindi possibile salvare tutti gli oggetti in un colpo solo chiamando `file.Write();` (riga 32).
NOTA: se l'istogramma fosse stato dichiarato nella stack, per salvarlo nel file sarebbe stato necessario utilizzare:

```
hist.DrawCopy();
```

il comando `DrawCopy()` infatti (a differenza del comando `Draw()`) crea una copia dell'oggetto nella heap che a questo punto sopravvive all'uscita dallo scope.
La macro creata viene caricata ed eseguita tramite:

```
root[ ]  .L ioexbis.C
root[ ]  ioexbis("dati0.dat","output.root")
```

5 Codici

5.1 ioexample2.C

```
#include <Riostream.h>
#include <TH1D.h>
#include <TFile.h>
#include <TCanvas.h>
using namespace std;

void ioexample2(const string& fimpName, const string& histName)
{
    ifstream in(fimpName);
    if (!in)
    {
        cout << "Il file " << fimpName << " non esiste." << endl;
        return;
    }

    TH1D* hist;
    hist = new TH1D("hist", "Istogramma", 100, -7., 7.);

    int count = 0;
    double x;
    while(in >> x)
    {
        count++;
        hist->Fill(x);
    }

    cout << "Dati letti: " << count << endl;
    in.close();
    hist->Draw();

    TFile file(histName.c_str(), "recreate");
    hist->Write();
    file.Close();
}
```

5.2 ioexbis.C

```
#include <Riostream.h>
#include <TH1D.h>
#include <TFile.h>
#include <TCanvas.h>
using namespace std;

void ioexbis(const string& fimpName, const string& histName)
{
    ifstream in(fimpName);
    if (!in)
    {
        cout << "Il file " << fimpName << " non esiste." << endl;
        return;
    }

    TFile file(histName.c_str(), "recreate");

    TH1D* hist;
    hist = new TH1D("hist", "Istogramma", 100, -7., 7.);

    int count = 0;
    double x;
    while(in >> x)
    {
        count++;
        hist->Fill(x);
    }

    cout << "Dati letti: " << count << endl;
    in.close();
    hist->Draw();
    file.Write();

    file.Close();
}
```