ROMA TRE
UNIVERSITÀ DEGLI STUDI

UNIVERSITA' DEGLI STUDI ROMA TRE

DIPARTIMENTO DI FISICA "E. AMALDI"

Laboratorio di Programmazione e Calcolo

6 crediti

a cura di

Severino Bussino

Anno Accademico 2021-22

0) Struttura del Corso

1) Trattamento dell'informazione
Elementi di Architettura di un Computer

Verra' trattata in una delle prossime lezioni

2) Sistemi operativi

3) Introduzione alla
Programmazione ad oggetti (OO)

4) Simulazione del Sistema Solare

5) Introduzione al linguaggio C/C++

6) Elementi di linguaggio C/C++

A 1 - istruzioni e operatori booleani

2 - iterazioni (`for`, `while`, `do … while`)

B - istruzioni di selezione (`if`, `switch`, `else`)

C - funzioni predefinite. La classe `math`.

7) Puntatori

8) Vettori (Array)

9) Vettori e Puntatori

10) Classe SistemaSolare (prima parte)

11) Gestione dinamica della memoria

12) Classe SistemaSolare

13) Programma di Simulazione (main)

Valutazione del Corso

ezioni | Calendario delle Esercitazioni | Materiale Didattico | Prove di Valutazione

You are here: Home › Prove di Valutazione › Test a risposte multiple

Test a risposte multiple

Un test di esonero a risposte multiple si svolgera'

Venerdi' 12 novembre 2021 in aula M3

dalle 11:00 alle 13:00

(Largo San Leonardo Murialdo, 1)

14) Ereditarieta'

- Concetto
- Funzione
- Realizzazione in C++
- Esempi

15) Classe Sonda

16) Output su file

14) Ereditarieta'

- Concetto
- Funzione
- Realizzazione in C++
- Esempi

Caratteristiche Fondamentali della Programmazione ad Oggetti (OO) (1) (dalla seconda lez.)

1. Incapsulamento

   I dati dell'oggetto sono nascosti ad altri oggetti ed è possibile accedervi solo attraverso modalita' ben definite

Robustezza Flessibilita'

1. Robustezza
2. Possibilita' di ri-utilizzo del codice
3. Portabilita'
4. Flessibilita' e Organizzazione del Codice (Semplicita')

Caratteristiche Fondamentali della Programmazione ad Oggetti (OO) (2) (dalla seconda lez.)

2. Ereditarieta’

   Gli oggetti complessi possono essere costruiti a partire da oggetti più semplici. Gli oggetti complessi derivano tutto o parte del loro comportamento dagli oggetti a partire dai quali sono stati generati

→ Ri-utilizzo del codice
Organizzazione del codice

1. Robustezza
2. Possibilita’ di ri-utilizzo del codice
3. Portabilita’
4. Flessibilita' e Organizzazione del Codice (Semplicita')

Caratteristiche Fondamentali della Programmazione ad Oggetti (OO) (3) (dalla seconda lez.)

3. Polimorfismo

   Oggetti *simili* possono essere trattati, in alcuni casi, come se fossero dello stesso tipo, senza la necessità di implementare trattamenti specifici per distinguere tra le varie tipologie

→ Ereditarieta' piu' efficiente   Flessibilita'

La Portabilita' e' una caratteristica del C++ standard (ANSi-C++)

1. Robustezza
2. Possibilita' di ri-utilizzo del codice
3. Portabilita'
4. Flessibilita' e Organizzazione del Codice (Semplicita')

Le Classi nella Programmazione ad Oggetti

(dalla seconda lez.)

1. Incapsulamento ←
2. Ereditarieta'
3. Polimorfismo

Incapsulamento → Oggetti

Oggetti → Classi

~~Prossima lezione~~ terza lezione

Classi in OO → Come scrivere una classe in C++

L'Ereditarieta' nella Programmazione ad Oggetti

1. Incapsulamento
2. Ereditarieta' ←
3. Polimorfismo

Riutilizzo del Codice → Ereditarieta'

Relazioni tra concetti → Ereditarieta'

~~Prossima lezione~~ *oggi*

Ereditarieta' in OO → Come implementare l'Ereditarieta' in C++

Premessa

Relazioni tra Concetti ed Ereditarieta'

«A concept does not exist in isolation. It coexists with related concepts and derives much of its power from relationships with related concepts…» (*B. Stroustrup*)

Indipendenza tra Concetti e Programmazione Generica

«Independent concepts should be independently represented and should be combined only when needed. Where this principle is violated, you either bundle unrelated concepts together or create unnecessary dependencies. Either way, you get a less flexible set of components out of which to compose systems…» (*B. Stroustrup*)

Relazioni tra concetti Ereditarieta' *oggi*

Concetti Indipendenti *Templates* *Cenni in una delle ultime lezioni*

“Indice” (1)

0. Aspetti generali sull’Ereditarieta’ (**Obiettivi**)
    - Concetti fondamentali
    - Utilizzo
    - Ereditarieta’ e ri-utilizzo del codice

1. Ereditarieta’ in C++ (prima parte)
    - Il meccanismo dell’ereditarieta’
    - `public` `private` `protected`

“Indice” (2)

2. Ereditarieta' per esempi
   - La classe Sonda
   - Dichiarazione della classe Sonda (Sonda.h)
   - Implementazione della classe Sonda (Sonda.cc)

3. Ereditarieta' in C++ (seconda parte)
   - Ereditarieta' e puntatori
   - Ereditarieta' multipla (virtual inheritance)
   - Metodi virtuali (ereditarieta' e polimorfismo)
   - Metodi *pure virtual* e classi astratte

} *prossima lezione*

Aspetti generali sull'Ereditarieta' (I)

1. Concetti fondamentali
   - «A concept does not exist in isolation. It coexists with related concepts and derives much of its power from relationships with related concepts…» (*B. Stroustrup*)
   - Distinguere composizione da ereditarieta'
   - Esempi: automobile, figure piane, dipendente, ...

Aspetti generali sull'Ereditarieta' (II)

2. Ereditarieta' e ri-utilizzo del codice

   - L'ereditarieta' permette di ri-utilizzare il codice e quindi di rendere piu' "economica" la gestione di programmi complessi

   - Codici anche complessi sono piu' comprensibili e meglio organizzati

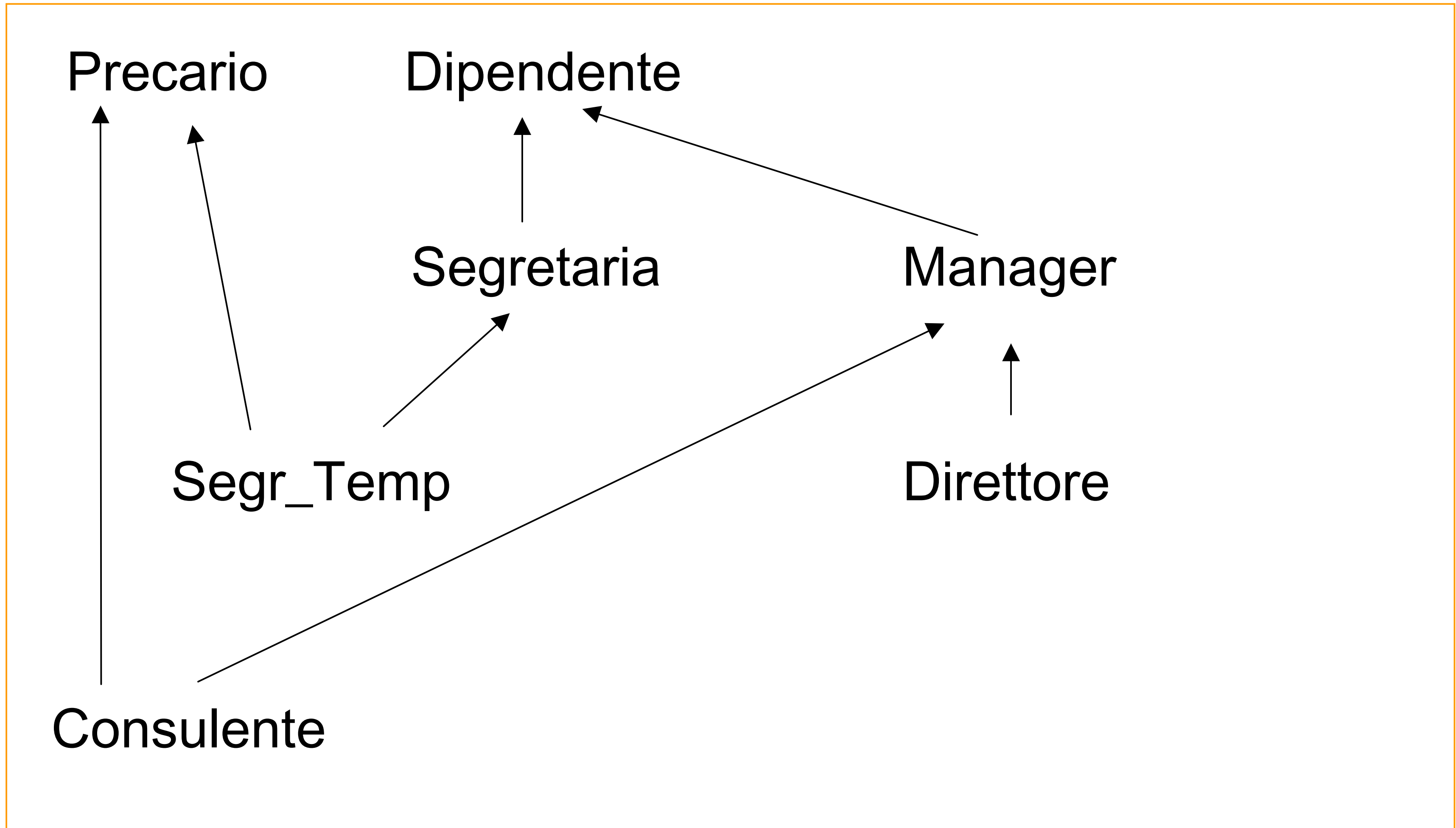
   - Le relazioni tra concetti appaiono nel codice come relazioni di ereditarieta'

Linguaggio Naturale	⟷	Programmazione ad Oggetti
Relazione tra concetti	⟷	Ereditarieta'

Aspetti generali sull'Ereditarieta' (III)

3. Utilizzo
   - Si identificano classi che appaiono in relazione
   - Le classi "in relazione" si organizzano gerarchicamente
   - Tra classi organizzate gerarchicamente si identificano
     - nuovi attributi
     - nuovi metodi
     - vecchi metodi che devono essere ri-definiti

Aspetti generali sull'Ereditarieta' (IV)



Ereditarieta' in C++ (prima parte) (1)

1. Il meccanismo dell'ereditarieta'

```
class Sonda : public CorpoCeleste
 {
   private:
    .......................................
   protected:
    .......................................
   public:
    .......................................

 };
```

Ereditarieta' in C++ (prima parte) (2)

2. `public` `private` `protected`

Tipo di Ereditarieta'	**Attributi e Metodi Pubblici**	**Attributi e Metodi Protected**	**Attributi e Metodi Privati**
`public`	Pubblici	Protected	non accessibili
`protected`	Protected	Protected	non accessibili
`private`	Privati	Privati	non accessibili

Ricordando la Struttura ……………

Ereditarieta' per esempi: la classe Sonda

CorpoCeleste

Nome (stringa)
m (num. reale)
x (num. reale)
y (num. reale)
vx (num. reale)
vy (num. reale)

CorpoCeleste(….)
~Corpoceleste()
CalcolaPosizione(forza, dt)
StampaVelocita'()
StampaPosizione()
M()
X() Y()
Vx() Vy()

Sonda

tCount (num. reale)
tStart (num. reale)
owner (CorpoCeleste *)
svx (num. reale)
svy (num. reale)
started (stringa)

Sonda(…)
~Sonda()
CalcolaPosizione(forza, dt)

Sonda.h

Sonda.cc

Sonda.h

```
#ifndef SONDA_H
#define SONDA_H

#include "CorpoCeleste.h"

class Sonda : public CorpoCeleste {

 protected:
    float tCount;
    float tStart;
    CorpoCeleste *owner;
    float svx;
    float svy;
    char started;

 public:
    Sonda() { } ;
    Sonda(string name, float mass,float starttime,
            CorpoCeleste *startFrom, float vxi, float vyi);

    ~Sonda() { } ;

    void calcolaPosizione(float fx, float fy, float t);

};

#endif
```

Sonda.cc (1)

```
#include "Sonda.h"
#include <string>
#include <iostream>

using namespace std ;

Sonda::Sonda(string name, float mass, float starttime,
     CorpoCeleste *startFrom, float vxi,
     float vyi) : CorpoCeleste(name, mass, 0., 0., 0., 0.) {

    tStart = starttime ;
    tCount = 0         ;
    owner = startFrom  ;
    svx = vxi        ;
    svy = vyi        ;
    started = 0      ;
    x = owner->X()   ;
    y = owner->Y()   ;
    vx = owner->Vx() ;
    vy = owner->Vy() ;
}
```

Sonda.cc (2)

```
void Sonda::calcolaPosizione(float fx, float fy, float t) {

  if (tCount<tStart) {
     x  = owner->X()  ;
     y  = owner->Y()  ;
     vx = owner->Vx() ;
     vy = owner->Vy() ;

  } else {

     if (!started) {
        cerr << "Sonda in partenza... " << endl ;
        vx += svx ;
        vy += svy ;
        started = 1 ;
     }
     CorpoCeleste::calcolaPosizione(fx,fy,t);
  }

  tCount += t ;

}
```

Ereditarieta' in C++ (seconda parte)

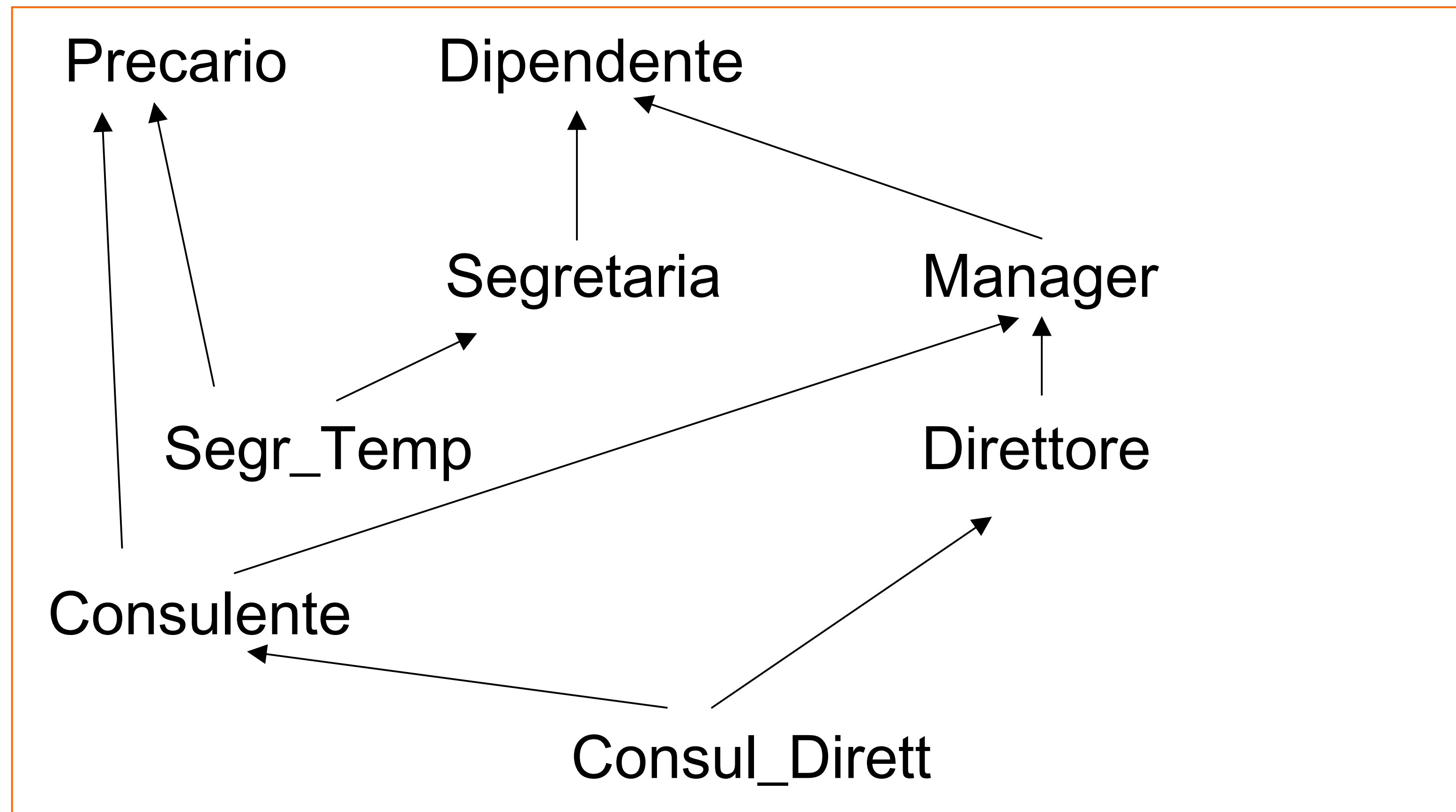
1. Ereditarieta' multipla (*virtual inheritance*)
2. Ereditarieta' e puntatori in C++
3. Metodi virtuale (ereditarieta' e polimorfismo)
4. Distruttori virtuali
5. Metodi *pure virtual* e classi astratte

Prossima lezione

Ereditarieta' multipla (*virtual inheritance*) (1)

1. Ereditarieta' multipla

2. Catena di ereditarieta' multipla



Ereditarieta' multipla (*virtual inheritance*) (2)

3. L'ereditarieta' *virtual* permette di evitare doppie coppie di una classe

```
class Direttore : virtual public Manager {

} ;

class Consulente : virtual public Manager {

} ;
```

Ereditarieta' e Puntatori in C++ (1)

1. Voglio rappresentare un Oggetto

```
CorpoCeleste terra(…..)   ;
```

2. Voglio rappresentare piu' oggetti

```
CorpoCeleste terra(…..)   ;
CorpoCeleste sole(…..)    ;
CorpoCeleste terra(…..)   ;
```

3. Voglio rappresentare tanti oggetti

```
CorpoCeleste pianeti[10]   ;
 pianeti[0] =  ….  ;
 pianeti[1] =  …   ;
 ……………..
```

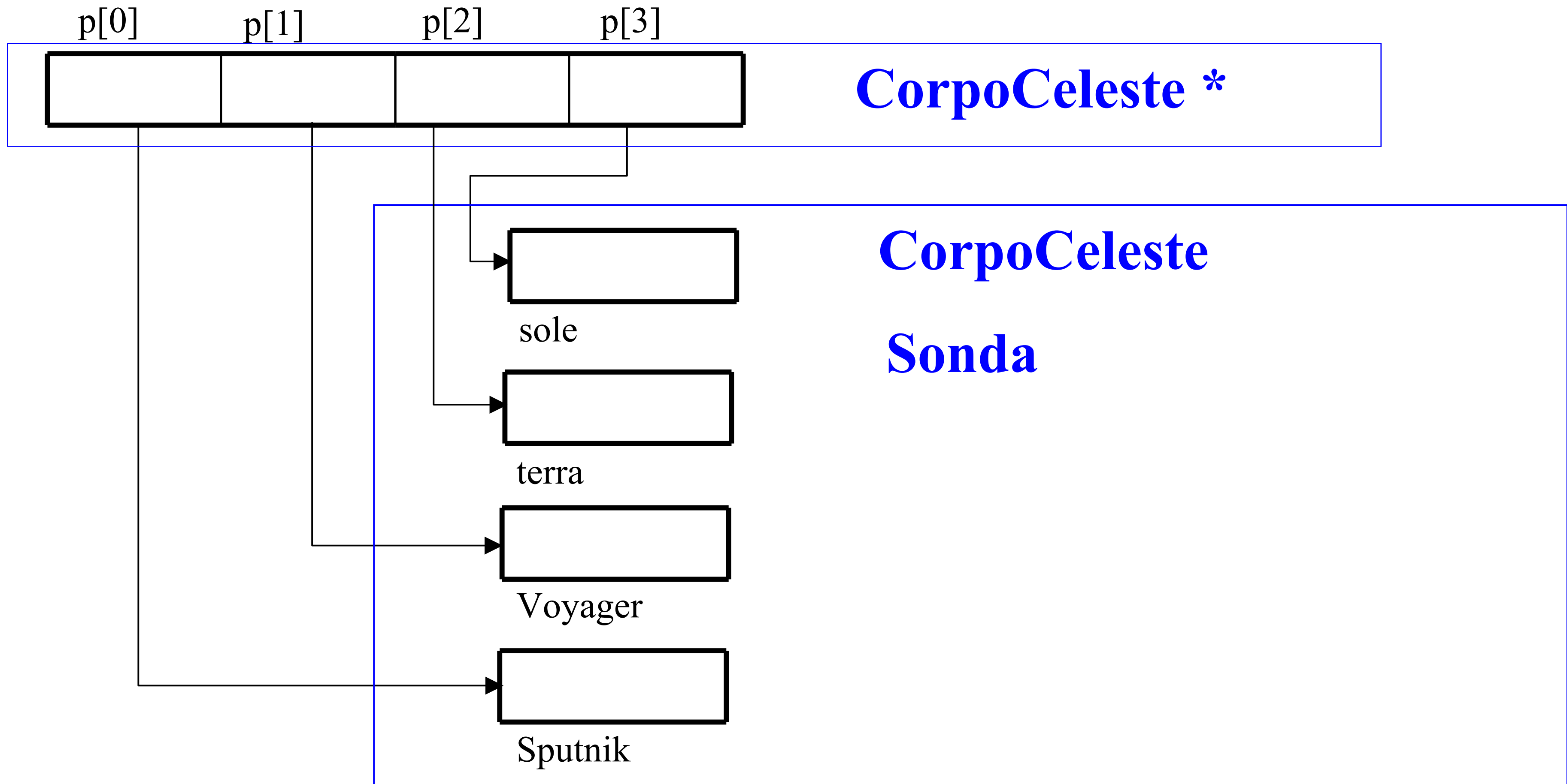
Ereditarieta' e Puntatori in C++ (2)

2. Voglio rappresentare tanti oggetti legati tra loro da ereditarieta'

devo usare un vettore di puntatori

```
CorpoCeleste * pianeti[10]         ;
pianeti[0] =  new CorpoCeleste(…) ;
pianeti[1] =  new Sonda(…)         ;
pianeti[2] =  new Corpoceleste(…) ;
………………………………………………………..
```

Se ho un vettore di puntatori ad oggetti CorpoCeleste…



… quale metodo `calcolaPosizione` viene applicato agli elementi del vettore?

Lo vedremo nella prossima lezione!

16) Input/Output su file

I/O su file: come accedere ad un file

1. Aprire un file
   - nome
   - Tipo (cioe' il modo in cui e' stato scritto o si vuole scrivere)

2. Leggere da un file

3. Scrivere su un file

4. Ottenere delle informazioni sul file
   - Esiste?
   - E' finito? (sto cercando di leggere oltre la fine?)
   - Quanto e' grande?

5. Chiudere un file

I/O su file: Aprire un file (1)

0. L'header file da includere e' <fstream>

```
#include <fstream>
```

1. Nome
   - In sola lettura

```
ifstream inFile("filename") ;
```

   - In sola scrittura

```
ofstream outFile("filename") ;
```

   - In lettura e scrittura (**sconsigliato**)

```
fstream ioFile("filename") ;
```

I/O su file: Aprire un file (2)

2. Se si usa una stringa C++ e' necessaria la conversione in stringa C. Ad esempio:

```
string Myfile= "filename.dat"    ;
ifstream inFile(Myfile.c_str()) ;
```

3. Tipo

Nella maggior parte dei casi, e' sufficiente utilizzare la modalita' testo, che e' quella di default. Per altre scelte, si veda un manuale o la pagina web relativa a fstream

Quando il file viene aperto, il programma si posiziona all'inizio del file. Per altre scelte, si veda un manuale o la pagina web relativa a fstream

I/O su file: leggere e scrivere su un file

Si usa come cout o cin !!!

- Scrivere su un file `<<`
  (ad esempio per scrivere le variabili a e b , separate da uno spazio, e poi andare a capo)

```
outFile << a << " " << b << endl ;
```

- Leggere da un file `>>`
  (ad esempio per leggere le variabili a e b)

```
inFile >> a >> b ;
```

Per leggere un file e' necessario sapere come e' stato scritto!

I/O su file: Ottenere delle informazioni sul file

- Tutto bene? Posso usare lo streaming associato al file?

falso(=0) Operazione non riuscita
vero (!=0) Operazione andata a buon fine

```
if(!inFile) {
  cerr << "Errore nell'apertura del file!" << endl ;
}
```

Oppure (ma attenzione a non fare if troppo lunghi...)

```
if(inFile) {

      ……………   // operazioni con il file

}
```

- E' finito? (sto cercando di leggere oltre la fine?)

  Si usa il metodo `eof()` applicato allo streaming associato al file

  falso(=0) Il file non e' finito - posso continuare a leggere

  vero (!=0) Sono alla fine del file
  - non posso continuare a leggere
  - l'ultima lettura non e' andata a buon fine

```
if(!inFile.eof()) {
      ..............    // posso continuare a leggere
}
..............
inFile >> a ;
if(!inFile.eof()) {
      ..............   // posso usare  a
}
```

`inFile.eof()` e' vero quando ci si trova **subito dopo** la fine del file

I/O su file: Chiudere un file

```
inFile.close()  ;
outFile.close() ;
ioFile.close()  ;
```

In sintesi

- Apro il file creando un oggetto di tipo ifstream o ofstream
- Uso `<<` e `>>` come se utilizzassi `cin` o `cout`
- Devo ricordarmi di controllare di non leggere oltre il file (ovvio!)
- Chiudo il file quando non mi serve piu'

I/O su file: un esempio (1)

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

using namespace std;

int main() {

  int imax = 100 ;
  int jmax = 3   ;

  string fileout = "numeri.dat" ;       //nome del file
  ofstream outFile(fileout.c_str()) ; //apro il file in scrittura

  for (int i=0; i<imax; i++) {          //scrivo sul file
     for (int j=0; j<jmax; j++){
        outFile << " " << i*10+j ;
     }
     outFile << endl ;
  }

  outFile.close();                  //chiudo il file - continua
```

```
 cout << endl << " ora leggo dal file " << fileout << endl ;

 ifstream inFile(fileout.c_str()) ;  // apro il file in lettura

 int a, b, c ;
 while(!inFile.eof()) {           // ciclo sino alla fine del file
                                  // leggo il file e ne scrivo
    cout << "    ";               // il contenuto su video
    for (int j=0; j<jmax; j++){
       inFile >>  a ;
       if(!inFile.eof()) {        // sono alla fine del file ?
           cout << " " << a  ;
       }                          // fine if
    }                             // fine for
    cout << endl;
 }                                // fine while

 outFile.close();                 // chiudo il file

return 1;

}
```

I/O su file: un esempio (2)

Come e' fatto il file:

```
nbseve(~)>cat numeri.dat
 0 1 2
 10 11 12
 20 21 22
 30 31 32
 40 41 42
 50 51 52
 60 61 62
 ......................................
 ......................................
 ......................................
 930 931 932
 940 941 942
 950 951 952
 960 961 962
 970 971 972
 980 981 982
 990 991 992
```

L'output del programma:

```
nbseve(~)>./prov_fs

 ora leggo dal file numeri.dat
 0 1 2
 10 11 12
 20 21 22
 30 31 32
 40 41 42
 50 51 52
 60 61 62
 ......................................
 ......................................
 ......................................
 930 931 932
 940 941 942
 950 951 952
 960 961 962
 970 971 972
 980 981 982
 990 991 992
```