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)

14) Ereditarieta'

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

15) Classe Sonda

16) Output su file

17) Polimorfismo

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

18) "per referenza" e "per valore"

19) Richiami sulle funzioni

20) L'algebra di una classe (cenni)

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)

Le lezioni in aula si concludono il 3 dicembre

Venerdi' 26 novembre NON ci sara' lezione

Le Esercitazioni di Laboratorio proseguono
senza variazioni con l'orario gia' comunicato
(e pubblicato sul sito)

16 nov - 17 nov - 18 nov	Classe Shape (Ereditarieta')
23 nov - 24 nov - 25 nov	Classe Shape (sec. parte se necessario ...)
30 nov - 1 dic - 2 dic	Classe Sonda + container STL per i pianeti
14 - 15 - 16 dicembre	Simulazione Prova Individuale

11 - 12 - 13 gennaio Prova Individuale

Osservazioni sulla Esercitazione di Laboratorio n. 5 (Array e Puntatori) (0)

Obiettivi

- Come si utilizza un vettore di Oggetti (`double` o `CorpoCeleste`)
- Come si scorre un vettore
  - con un indice
  - con un puntatore
- Come si accede agli elementi di un vettore
  - con un indice
  - con un puntatore
- Come si utilizza un vettore di puntatori ad Oggetti (`double*` o `CorpoCeleste*`)
- Ho verificato l'*out of scope*?
- Come si usa l'istruzione `new` ? Perche' si usa? (e non dimenticare l'istruzione `delete`)

Come si utilizza un vettore di Oggetti
(double **o** CorpoCeleste)

```
#include <iostream>
#include <sstream>
#include <string>
#include "CorpoCeleste.h"

using namespace std;


int main() {


  // istanzio un vettore di double e di oggetti di tipo CorpoCeleste
  // riempio i due vettori e poi ne stampo (in parte) il contenuto

  double dvect[5]          ;
  CorpoCeleste pianeti[5] ;


  CorpoCeleste marte ("Marte",     1.,  2.,  3.,  4.,  5.) ;
  CorpoCeleste terra ("Terra",    11., 12., 13., 14., 15.) ;
  CorpoCeleste giove ("Giove",    21., 22., 23., 24., 25.) ;
  CorpoCeleste pluto ("Plutone", 31., 32., 33., 34., 35.) ;
  CorpoCeleste sole  ("Sole",     41., 42., 43., 44., 45.) ;
```


segue...

```
// riempio i vettori

for (int i=0; i<5; i++) {

   dvect[i]   = (double) i ;
}

pianeti[0] = marte ;
pianeti[1] = terra ;
pianeti[2] = giove ;
pianeti[3] = pluto ;
pianeti[4] = sole  ;


// oppure
// diverso modo di inizializzare
// per il resto e' equivalente al precedente


 double dvec2[5] = {0.,1.,2.,3.,4.} ;

 CorpoCeleste pianet2[5] = {marte, terra, giove, pluto, sole} ;
```


segue...

Come si scorre un vettore

...continua

```
// con un indice


 cout << endl ;

  for (int i=0; i<5; i++) {

    cout << " dvect[" << i << "] = "            << dvect[i]
    << endl ;
    cout << " nome  di pianeti["<< i << "] = " << pianeti[i].nome()
    << endl ;
    cout << " massa di pianeti["<< i << "] = " << pianeti[i].M()
    << " Kg " << endl ;

    cout << endl ;
  }


// con un puntatore
```


segue...

...continua

```
// con un puntatore
  int i = 0 ;
  for (double * q = dvect ; q <= &dvect[4] ; q++ ) {
    cout << " dvect[" << i << "] = ";
    cout <<                          dvect[i++] << "  " ;
    cout << " q = "   << q << endl ;
  }

  cout << endl ;
  i = 0 ;

  for (CorpoCeleste * q = pianeti ; q <= &pianeti[4] ; q++ ) {
    cout << " nome  di pianeti["<< i << "] = "  << pianeti[i].nome()
    << endl ;
    cout << " massa di pianeti["<< i << "] = "  ;
    cout                                        << pianeti[i++].M()
    << " Kg " << "   " ; ;
    cout << " q = "   << q << endl ;

  }
```


segue...

Come si accede agli elementi di un vettore

...continua

```
// con un indice
// gia' visto  dvect[i]   oppure      pianeti[i].nome()

// con un puntatore. Ora scorro i vettori (qui con int, ma potreste usare
// i puntatori) e stampo il loro contenuto utilizzando i puntatori


  cout << endl ;
  for (i=0 ; i <5 ; i++ ) {

    cout << " dvect[" << i << "] = " << *(dvect+i)
    << endl ;
    cout << " nome  di pianeti["<< i << "] = "  << (pianeti+i)->nome()
    << endl ;
    cout << " massa di pianeti["<< i << "] = "  << (pianeti+i)->M()
    << " Kg " << endl ;
    cout << endl ;

  }
```


segue...

```
// con un puntatore. Ora scorro i vettori (qui con int, ma potreste usare
// i puntatori) e stampo il loro contenuto utilizzando i puntatori
// oppure in maniera equivalente

  cout << endl ; cout << endl ;
  double       * pv = dvect;
  CorpoCeleste * pp = pianeti;

  for (i=0 ; i <5 ; i++ ) {

    cout << " dvect[" << i << "] = " << *pv++
    << endl ;
    cout << " nome  di pianeti["<< i << "] = "  << pp->nome()
    << endl ;
    cout << " massa di pianeti["<< i << "] = "  << pp++->M()   << " Kg
    " << endl ;
    cout << endl ;

  }
```


segue...

Come si utilizza un vettore di puntatori ad Oggetti

(double* o CorpoCeleste*)

...continua

```
// istanzio un vettore di puntatori a double e di puntatori a oggetti
// di tipo CorpoCeleste
// riempio i due vettori e poi ne stampo (in parte) il contenuto

 double * pvect[5]          ;
 CorpoCeleste * ppianet[5] ;

 double di ;
 for (i=0; i<5; i++) {

   di = i             ;   // conversione da intero a double
   pvect[i]   = &di ;   // per ottenere un double*
 }

 ppianet[0] = &marte ;
 ppianet[1] = &terra ;
 ppianet[2] = &giove ;
 ppianet[3] = &pluto ;
 ppianet[4] = &sole  ;
```


segue...

...continua

```
cout << endl ;

for (int i=0; i<5; i++) {

  cout << " pvect[" << i << "] punta a  "              << *pvect[i]
  << endl ;
  cout << " il puntatore pvect[" << i << "] =  "       <<  pvect[i]
  << endl ;
  cout << " nome  del pianeta a cui punta ppianet["<< i << "] = " <<
  ppianet[i]->nome() << endl ;
  cout << " massa del pianeta a cui punta ppianet["<< i << "] = " <<
  (*ppianet[i]).M()  << " Kg " << endl ;

  cout << endl ;
}
```

segue...

Ho verificato l'*out of scope*?

...continua

```
  // ora riempio i vettori creando l'oggetto all'interno del loop
  // e scopre che non lo posso stampare perche e' out of scope.
  // Il vettore e' stato riempito correttamente (per copia)

CorpoCeleste * ppiane4[5] ;
cout << endl ;

 for (int i=0; i<5; i++) {

    ostringstream os ;
    os << i  ;
    CorpoCeleste ausi("Ausiliario_"+os.str(), i,i,i,i,i);

    ppiane4[i] = &ausi ;

  }
```


segue...

...continua

```
cout << endl ;
cout << endl ;

for (int i=0; i<5; i++) {

// cout << " nome  del pianeta a cui punta ppiane4["<< i << "] = "
//      <<   ppiane4[i]->nome() << endl ;
cout << " massa del pianeta a cui punta ppiane4["<< i << "] = "
     << ppiane4[i]->M()    << " Kg " << endl ;
cout << " valore del puntatore ppiane4[" << i << "] = "
     << ppiane4[i]         << endl ;

cout << endl ;

}
```

segue...

Come si usa l'istruzione `new` ? Perche' si usa?
(e non dimenticare l'istruzione `delete`)

...continua

```
// Ora utilizzo new per risolvere i DUE problemi visti sino ad ora
// pvect[i] aveva, per ogni i, lo stesso indirizzo che puntava al valore 4
// Nel vettore di puntatori a pianeti, ausi va out of scope
  double       * pvec2[5]   ;
  CorpoCeleste * ppiane5[5] ;

  cout << endl ;

  for (int i=0; i<5; i++) {

    pvec2[i] = new double(i);

    ostringstream os ;
    os << i  ;

    CorpoCeleste * Ausi = new CorpoCeleste("Ausiliario", i,i,i,i,i);
    ppiane5[i]   = Ausi ;

  }
```

segue...

...continua

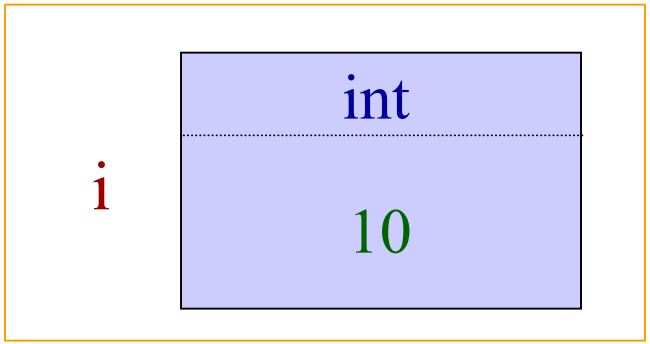
```
for (int i=0; i<5; i++) {
    cout << " pvec2[" << i << "] punta a  "    << *pvec2[i] << endl ;
    cout << " puntatore pvec2[" << i << "] = " <<  pvec2[i] << endl ;
    cout << " nome  del pianeta a cui punta ppiane5["<< i << "] = "
         << ppiane5[i]->nome() << endl ;
    cout << " massa del pianeta a cui punta ppiane5["<< i << "] = "
         << ppiane5[i]->M()    << " Kg " << endl ;
    cout << " valore del puntatore ppiane5[" << i << "] = "
         << ppiane5[i]         << endl ;
    cout << endl ;
  }
 // dobbiamo invocare delete su tutti gli oggetto costruiti con new

  for (int i = 0; i<5 ; i++ ) {

    delete pvec2[i]   ;
    delete ppiane5[i] ;

  }

  cout << endl ;

  return 1;
}
```

Osservazioni sulla Esercitazione di Laboratorio n. 5 (Array e Puntatori) (1)

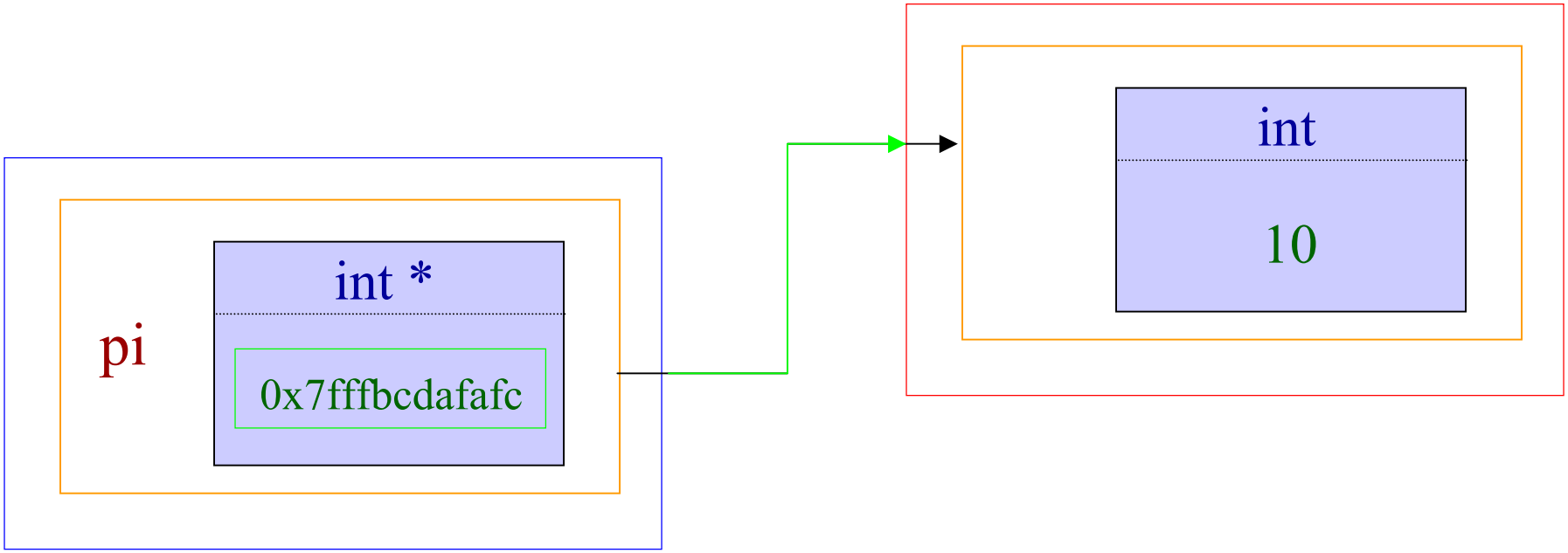
Premessa

Consideriamo la dichiarazione con inizializzazione:

```
int i =10 ;
```



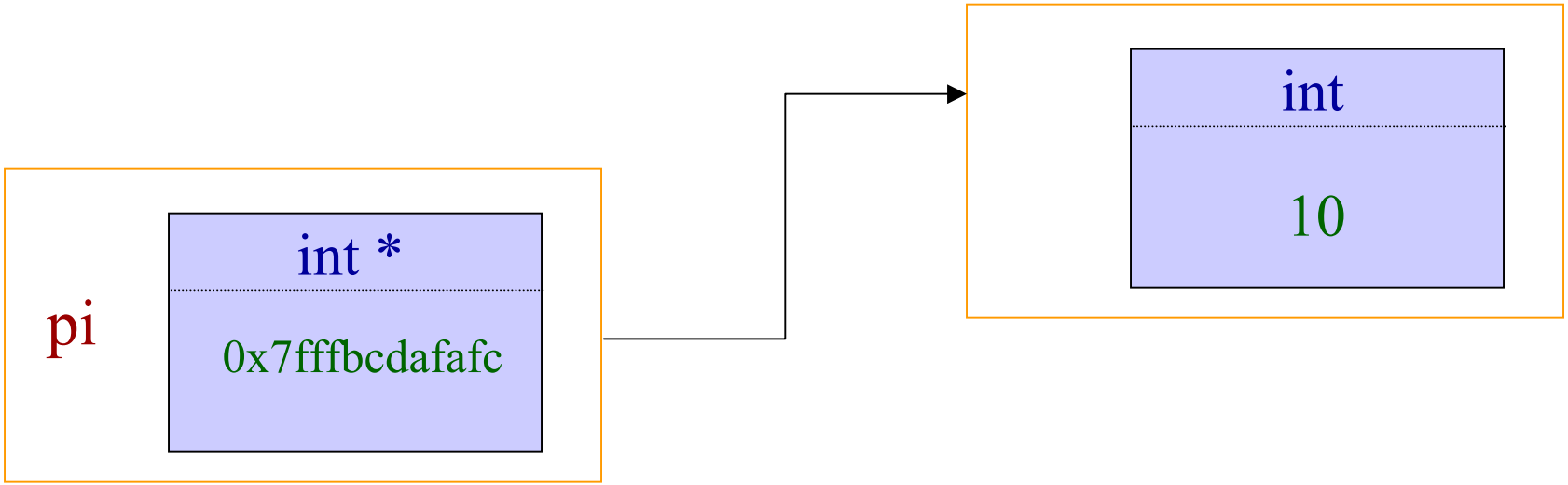
Osservazioni sulla Esercitazione di Laboratorio n. 5 (Array e Puntatori) (2)



Consideriamo la dichiarazione con inizializzazione:

```
int * pi = new int(10) ;
```

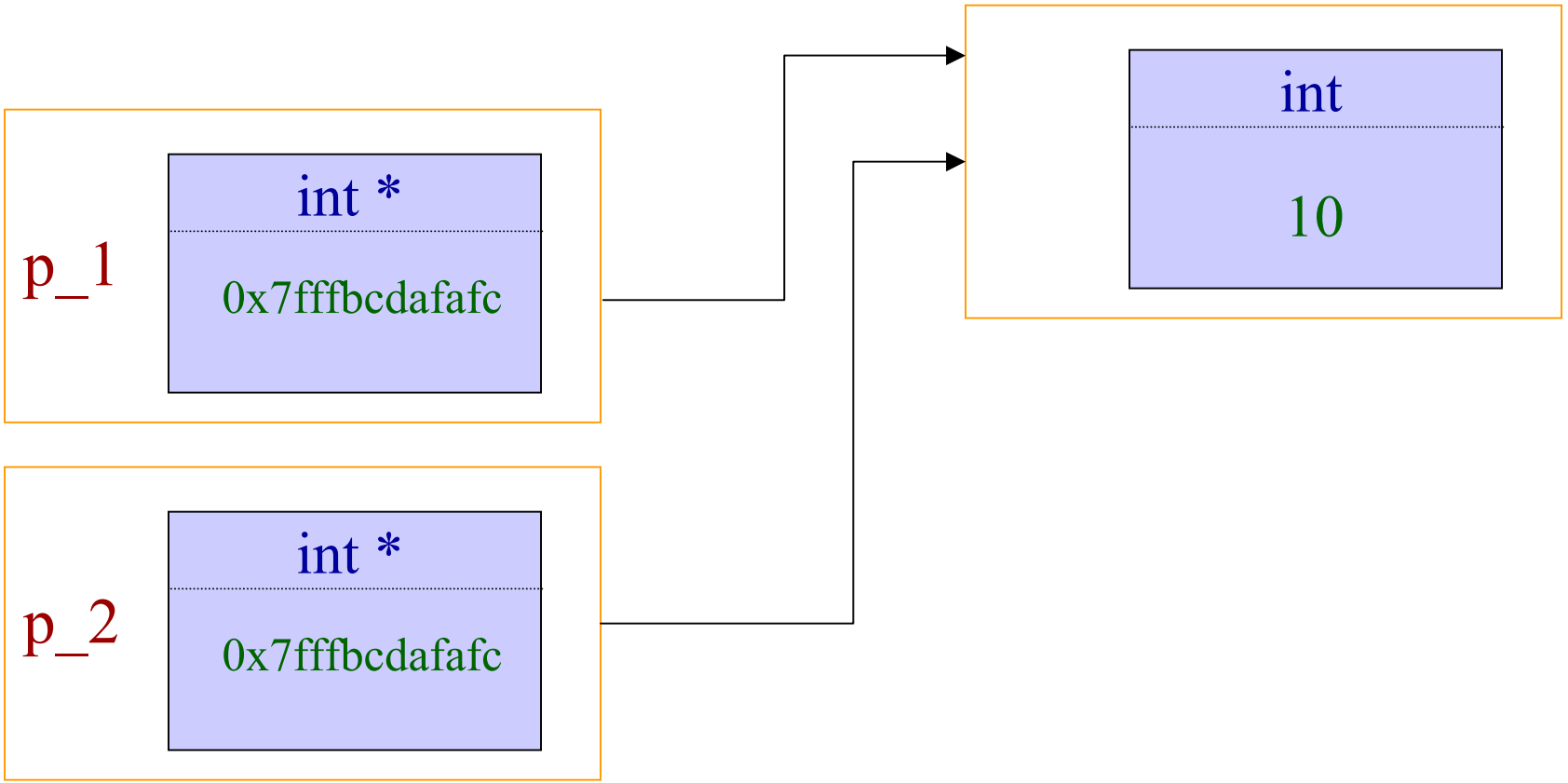
Osservazioni sulla Esercitazione di Laboratorio n. 5 (Array e Puntatori) (2)



Consideriamo la dichiarazione con inizializzazione:

```
int * pi    =    new int(10)    ;
```

Osservazioni sulla Esercitazione di Laboratorio n. 5 (Array e Puntatori) (3)




```
int * p_1   =   new int(10)   ;

int * p_2   =   p_1           ;
```

Osservazioni sulla Esercitazione di Laboratorio n. 5 (Array e Puntatori) (4)

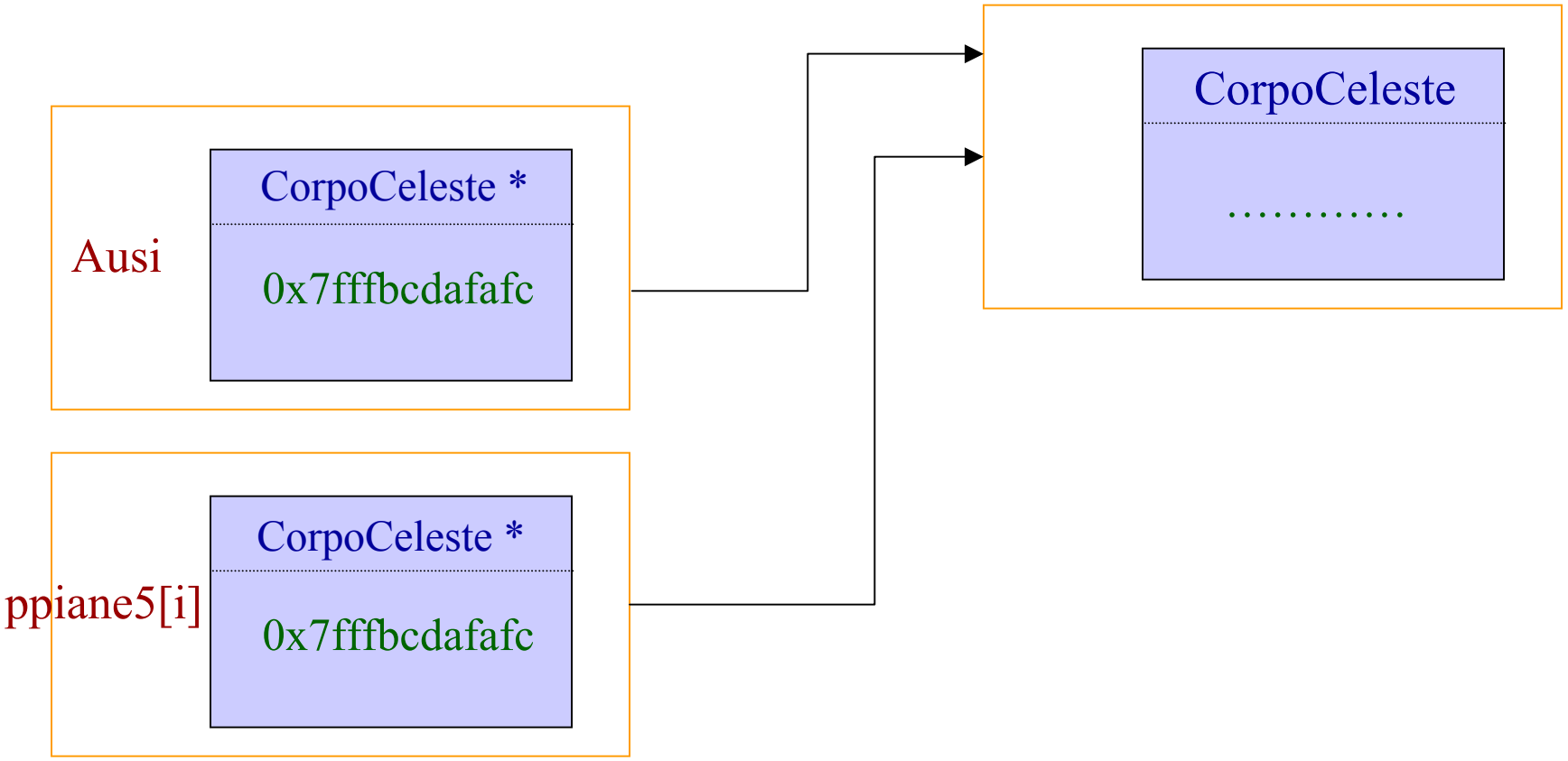
nbseve(~/Eserc_5)>g++ -c esercit_5_bis.cpp

esercit_5_bis.cpp: In function int main():

esercit_5_bis.cpp:426: error: Ausi was not declared in this scope

```
.....
double * pvec2[5]
CorpoCeleste * ppiane5[5] ;

for (int i=0; i<5; i++) {
  pvec2[i] = new double(i);
  CorpoCeleste * Ausi = new CorpoCeleste("Ausiliario", i,i,i,i,i);
  ppiane5[i]   = Ausi ;
}

cout << " Provo a stampare utilizzando Ausi " << endl;
cout << "   Ausi->M() = " << Ausi->M() << endl ;
```

Osservazioni sulla Esercitazione di Laboratorio n. 5 (Array e Puntatori) (5)



Quando esco dallo scope, viene cancellato Ausi, non l'oggetto a cui puntava, che e' ancora accessibile tramite ppiane5[i].

Osservazioni sulla Esercitazione di Laboratorio n. 5 (Array e Puntatori) (6)

```
// continuo dal precedente escluse le 2 ultime righe
cout << endl
     << " Ora stampo scorrendo i vettori pvec2[i] e ppiane5[i] " << endl;

for (int i=0; i<5; i++) {
   cout << " pvec2[" << i << "] punta a  "    << *pvec2[i] << endl ;
   cout << " puntatore pvec2[" << i << "] = " <<  pvec2[i] << endl ;

   cout << " nome  del pianeta a cui punta ppiane5["<< i << "] = "
        << ppiane5[i]->nome() << endl ;
   cout << " massa del pianeta a cui punta ppiane5["<< i << "] = "
        << ppiane5[i]->M()    << " Kg " << endl ;
   cout << " valore del puntatore ppiane5[" << i << "] = "
        << ppiane5[i]         << endl ;

   cout << endl ;
}
```

```
pvec2[0] punta a  0
puntatore pvec2[0] = 0xb701f0
nome  del pianeta a cui punta ppiane5[0] = Ausiliario
massa del pianeta a cui punta ppiane5[0] = 0 Kg
valore del puntatore ppiane5[0] = 0xb70210

pvec2[1] punta a  1
puntatore pvec2[1] = 0xb70250
nome  del pianeta a cui punta ppiane5[1] = Ausiliario
massa del pianeta a cui punta ppiane5[1] = 1 Kg
valore del puntatore ppiane5[1] = 0xb702a0

pvec2[2] punta a  2
puntatore pvec2[2] = 0xb702e0
nome  del pianeta a cui punta ppiane5[2] = Ausiliario
massa del pianeta a cui punta ppiane5[2] = 2 Kg
valore del puntatore ppiane5[2] = 0xb70330

pvec2[3] punta a  3
puntatore pvec2[3] = 0xb70370
nome  del pianeta a cui punta ppiane5[3] = Ausiliario
massa del pianeta a cui punta ppiane5[3] = 3 Kg
valore del puntatore ppiane5[3] = 0xb703c0

pvec2[4] punta a  4
puntatore pvec2[4] = 0xb70400
nome  del pianeta a cui punta ppiane5[4] = Ausiliario
massa del pianeta a cui punta ppiane5[4] = 4 Kg
valore del puntatore ppiane5[4] = 0xb706b0
```

Un'ultima osservazione.....

..... sarebbe meglio scrivere

```
  ..................................................
double * pvec2[5]          ;
CorpoCeleste * ppiane5[5] ;

for (int i=0; i<5; i++) {
  pvec2[i]   = new double(i);
  ppiane5[i] = new CorpoCeleste("Ausiliario", i,i,i,i,i);
}
```

17) Polimorfismo

- 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~~ *lezione scorsa*

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

Il Polimorfismo nella Programmazione ad Oggetti

1. Incapsulamento
2. Ereditarieta'
3. Polimorfismo ←

Flessibilita' ⟹ Polimorfismo

Ereditarieta' ⟺ Polimorfismo

~~Prossima lezione~~ *oggi*

Polimorfismo in OO ⟹ Come implementare il Polimorfismo in C++

"Indice"

0. Ereditarieta' e Polimorfismo

1. Metodi Virtuali

2. Metodi pure virtual e classi astratte

Il problema con cui abbiamo concluso la scorsa lezione…

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] =  …  ;
 ..................
```

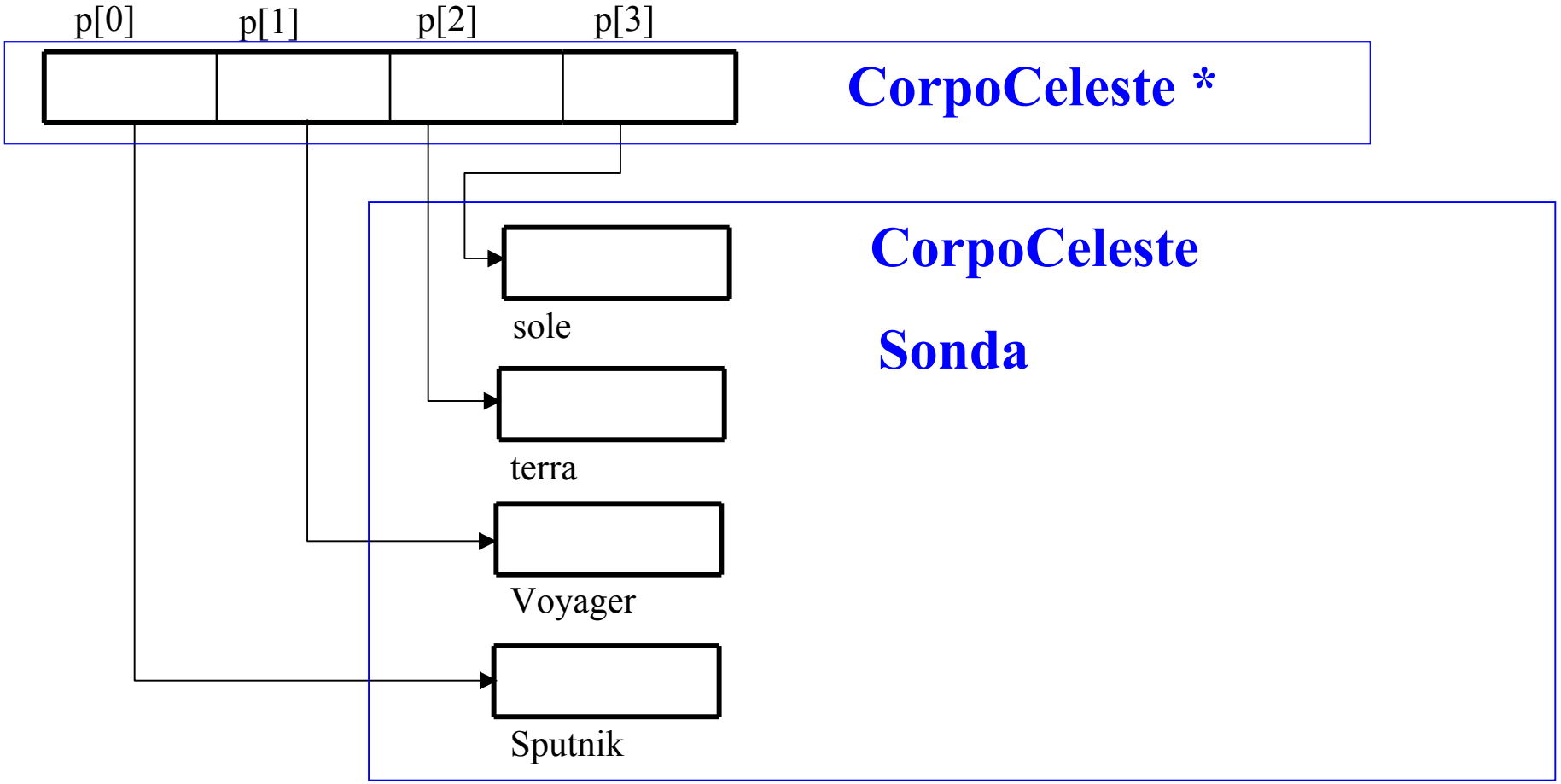
Ereditarieta' e Puntatori in C++ (2)

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

devo usare un vettore di puntatori

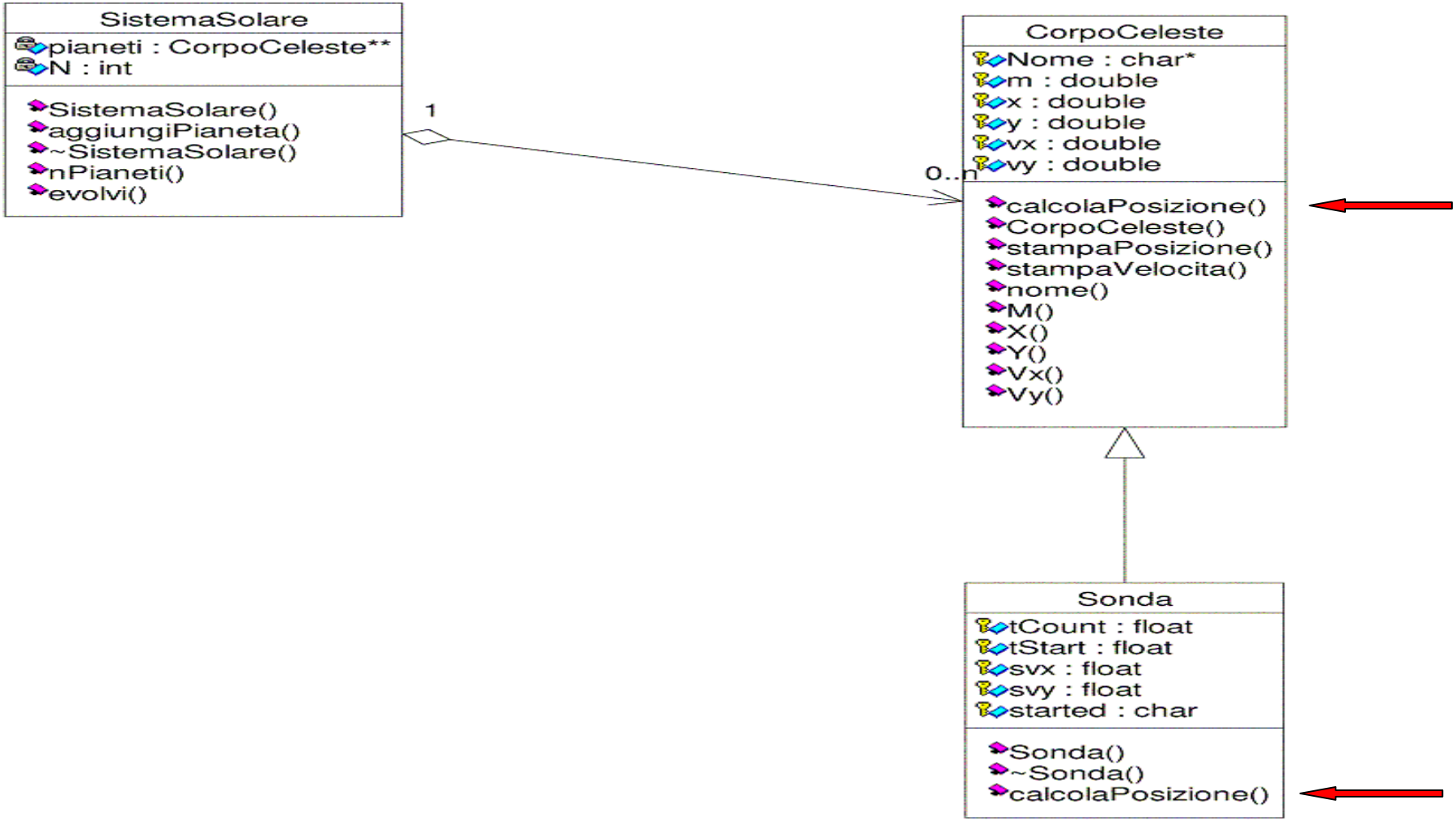
```
CorpoCeleste * pianeti[10]          ;
pianeti[0] =  new CorpoCeleste(…)  ;
pianeti[1] =  new Sonda(…)          ;
pianeti[2] =  new Corpoceleste(…)  ;
……..
```

Se ho un vettore di puntatori ad oggetti CorpoCeleste…



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

Ricordando la Struttura ……



Il problema non si porrebbe se usassimo gli oggetti…

```
…….

CorpoCeleste Terra("terra", ….) ;

Sonda Sputnik("sputnik", .…) ;

…….
Terra.calcolaPosizione(….) ;
            // applica il metodo di CorpoCeleste

Sputnik.calcolaPosizione(….) ;
            // applica il metodo di Sonda
```

… ma noi usiamo i puntatori per poter gestire insieme oggetti CorpoCeleste e Sonda!

```
…….

CorpoCeleste* Terra = new CorpoCeleste("terra", ….) ;

CorpoCeleste* Sputnik = new Sonda ("sputnik", .…)   ;

…….

Terra->calcolaPosizione(….) ;
                    //applica il metodo di CorpoCeleste

Sputnik->calcolaPosizione(….) ;
                    //quale metodo?
```

Metodi Virtuali (Ereditarieta' e Polimorfismo) (1)

1. Per indicare al compilatore che deve cercare il metodo nella classe piu' "bassa" nella catena gerarchica si usa la parola `virtual`

2. `virtual` va utilizzato nella classe piu' "alta" nella catena gerarchica delle classi che devono utilizzare il polimorfismo
(ma e' bene ripeterlo anche nelle classi "figlie")

Metodi Virtuali (Ereditarieta' e Polimorfismo) (2)

3. Se un metodo e' dichiarato `virtual`

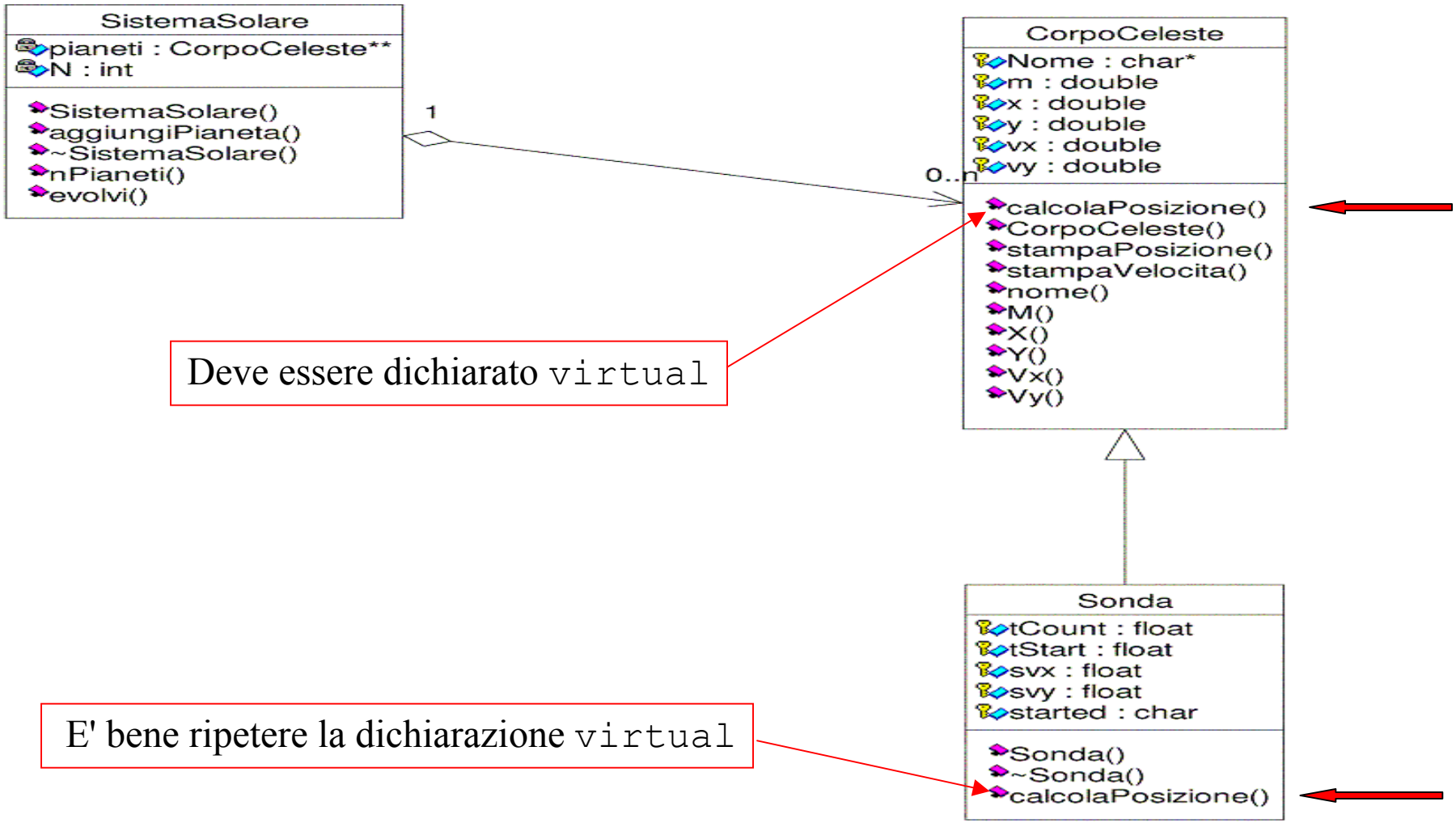
   - Il compilatore ricerca un metodo nella classe piu' bassa nella catena di ereditarieta' (esempio: `calcolaPosizione(fx,fy,dt)`)

   - Se non lo trova esegue il metodo nella classe piu' alta (esempio: `X()` `Vx()`)

4. Sintassi
   (nella definizione della classe - *header file* `.h`)

```
virtual void calcolaPosizione
        (float fx, float fy, float dt);
```

Ricordando la Struttura ……



CorpoCeleste.h (tenendo presenti le relazioni di ereditarieta')

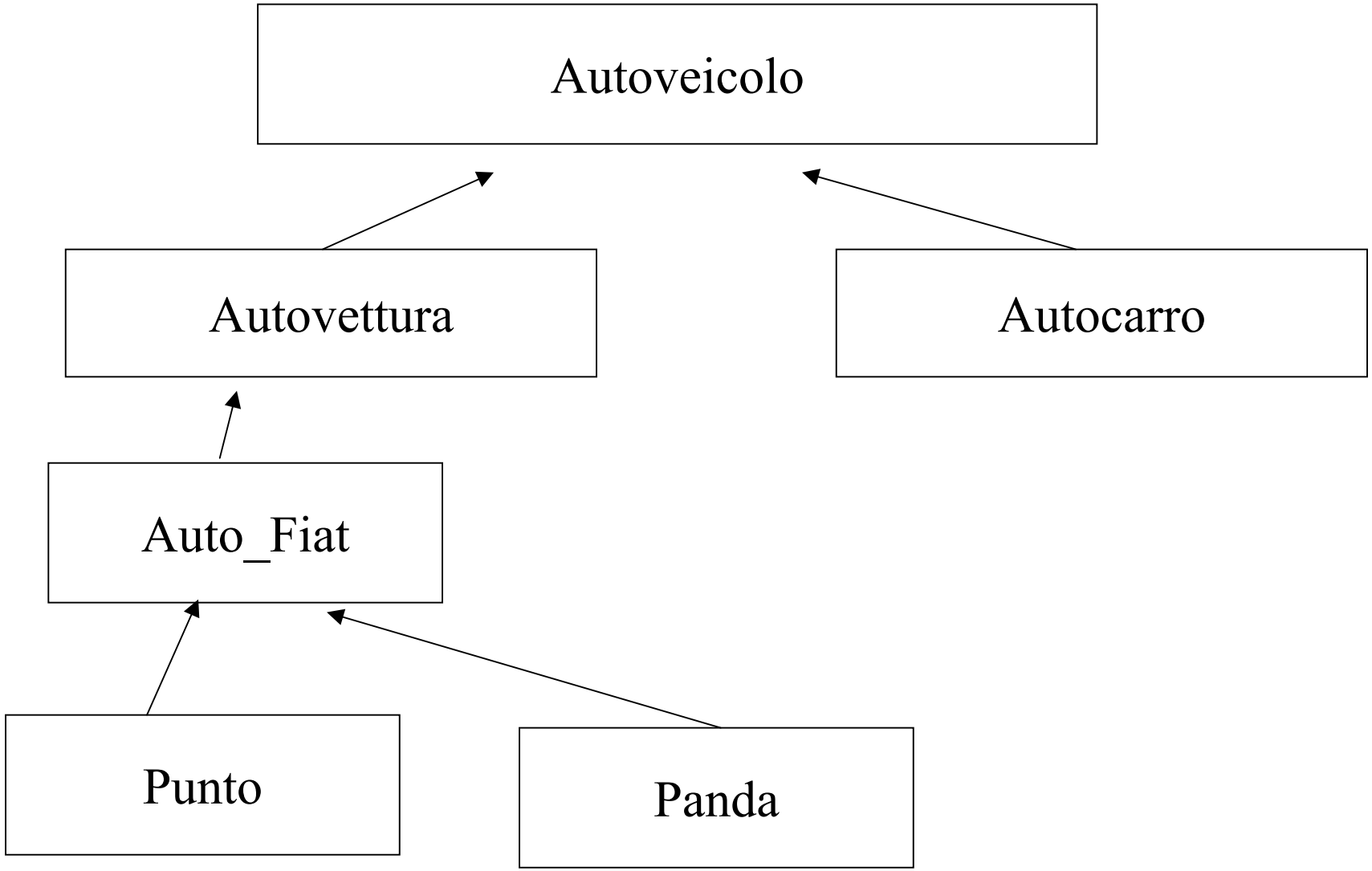
```
#ifndef  CORPOCELESTE_H
#define  CORPOCELESTE_H

#include <string>
#include <iostream>
#include <iomanip>

using namespace std;

class CorpoCeleste {
  protected:
        ..................

  public:
        ..................
        virtual void calcolaPosizione(float fx, float fy, float t);
        ..................

          .... (tutto il resto come prima … almeno per ora) ....

};

#endif
```

Sonda.h (modifica non necessaria ma **opportuna** per chiarezza)

```
#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() { } ;
    virtual void calcolaPosizione(float fx, float fy, float t);

};

#endif
```

.... nessun'altra modifica … almeno per ora ....

Un altro esempio



Definizione ed implementazione della classe Autoveicolo

Autoveicolo.h

```
#ifndef AUTOVEICOLO_H
#define AUTOVEICOLO_H

#include <iostream>
using namespace std;

class Autoveicolo {

 protected:

 public:

  Autoveicolo() { }
  ~Autoveicolo() { }

  virtual void ChiSei() ;

} ;

#endif
```

Autoveicolo.cc

```
#include "Autoveicolo.h"
#include <iostream>

using namespace std;

void Autoveicolo::ChiSei()  {

   cout <<
    " Sono un Autoveicolo!!! "
        << endl ;

}
```

Definizione ed implementazione della classe Autovettura

Autovettura.h

```
#ifndef AUTOVETTURA_H
#define AUTOVETTURA_H

#include "Autoveicolo.h"
#include <iostream>
using namespace std;

class Autovettura :
public Autoveicolo {

 protected:


 public:

  Autovettura() { }
  ~Autovettura() { }

  virtual void ChiSei() ;

} ;

#endif
```

Autovettura.cc

```
#include "Autovettura.h"
#include <iostream>

using namespace std;

void Autovettura::ChiSei() {

   cout <<
    " Sono una Autovettura!!! "
       << endl ;

}
```

Il problema non si porrebbe se usassimo (solo) gli oggetti…

```
#include "Autoveicolo.h"
#include "Autovettura.h"
#include <iostream>
using namespace std;

int main() {

  // proviamo con gli oggetti
  cout << endl;

  Autoveicolo primo ;
  Autovettura secondo ;

  primo.ChiSei() ;
  secondo.ChiSei() ;

  return 1;

}
```

```
nbseve(~/polim)>./prv_plom1

Sono un Autoveicolo!!!
Sono una Autovettura!!!
nbseve(~/polim)>
```

Lo stesso risultato si otterrebe senza dichiarare virtual il metodo `ChiSei()` in `Autoveicolo.h` e `Autovettura.h`

… ma noi usiamo i puntatori per poter gestire insieme …

```
#include "Autoveicolo.h"
#include "Autovettura.h"
#include <iostream>
using namespace std;

int main() {

  // proviamo con i puntatori
  Autoveicolo* veicolo[2] ;
  Autoveicolo primo    ;
  Autovettura secondo  ;

  veicolo[0] = &primo   ;
  veicolo[1] = &secondo ;

  veicolo[0]->ChiSei() ;
  veicolo[1]->ChiSei() ;

  return 1;

}
```

senza dichiarare virtual il metodo `ChiSei()`

```
nbseve(~/polim)>./prv_plom2
 Sono un Autoveicolo!!!
 Sono un Autoveicolo!!!
nbacer(~/polim)>
```

dichiarando virtual il metodo `ChiSei()`

```
nbseve(~/polim)>./prv_plom2
 Sono un Autoveicolo!!!
 Sono una Autovettura!!!
nbseve(~/polim)>
```

Ma anche i Costruttori e i Distruttori sono metodi!!!

I Costruttori (1)

```
#include "Autoveicolo.h"
#include "Autovettura.h"
#include <iostream>
using namespace std;

int main() {

  // proviamo con i puntatori
  Autoveicolo* veicolo[2] ;
  Autoveicolo primo     ;
  Autovettura secondo   ;

  cout << endl;
  veicolo[0] = new Autoveicolo() ;
  veicolo[1] = new Autovettura() ;

  return 1;

}
```

```
nbseve(~/polim)>./prv_plom3
 Sto creando un Autoveicolo...
 Sto creando un Autoveicolo...
 Sto creando una Autovettura...

 Sto creando un Autoveicolo...
 Sto creando un Autoveicolo...
 Sto creando una Autovettura...
nbacer(~/polim)>
```

I Costruttori (2)

O.K.!

Vedi ad esempio la dichiarazione della classe Sonda

```
nbseve(~/polim)>./prv_plom3
 Sto creando un Autoveicolo...
 Sto creando un Autoveicolo...
 Sto creando una Autovettura...

 Sto creando un Autoveicolo...
 Sto creando un Autoveicolo...
 Sto creando una Autovettura...
nbacer(~/polim)>
```

Quando viene invocato il **Costruttore** di una Classe che eredita da altre, il compilatore invoca automaticamente i Costruttori di tutte le Classi nella catena di ereditarieta', inziando dal costruttore **"piu' in alto"** nella catena gerarchica

I Distruttori (1) ….. se usassimo (solo) gli oggetti…

```
#include "Autoveicolo.h"
#include "Autovettura.h"
#include <iostream>
using namespace std;

int main() {

  // proviamo con gli oggetti

  Autoveicolo primo    ;
  Autovettura secondo  ;

  return 1;

}
```

```
nbacer(~/polim)>./prv_plom3
 Sto creando un Autoveicolo...
Sto creando un Autoveicolo...
Sto creando una Autovettura...
...sto distruggendo una Autovettura!!!
...sto distruggendo un Autoveicolo!!!
 ...sto distruggendo un Autoveicolo!!!
nbacer(~/polim)>
```

Quando viene invocato il **Distruttore** di una Classe che eredita da altre, il compilatore invoca automaticamente i Distruttori di tutte le Classi nella catena di ereditarieta', iniziando dal distruttore **"piu' in basso"** nella catena gerarchica

I Distruttori (2)

... ma noi usiamo i puntatori per poter gestire insieme ...

Se ad un oggetto si accede tramite il suo puntatore, istanziato come puntatore di una delle classi madre, **e' necessario** che il distruttore sia definito **`virtual`**, in modo che il compilatore inizi ad invocare il distruttore **piu' in basso** nella catena di ereditarieta', risalendo poi tutta la catena gerarchica.

I Distruttori Virtuali (3)

Autoveicolo.h

```
#ifndef AUTOVEICOLO_H
#define AUTOVEICOLO_H

#include <iostream>
using namespace std;

class Autoveicolo {

 protected:

 public:

  Autoveicolo() { … }
  virtual ~Autoveicolo()
    { ….    }

  virtual void ChiSei() ;

} ;

#endif
```

Autovettura.h

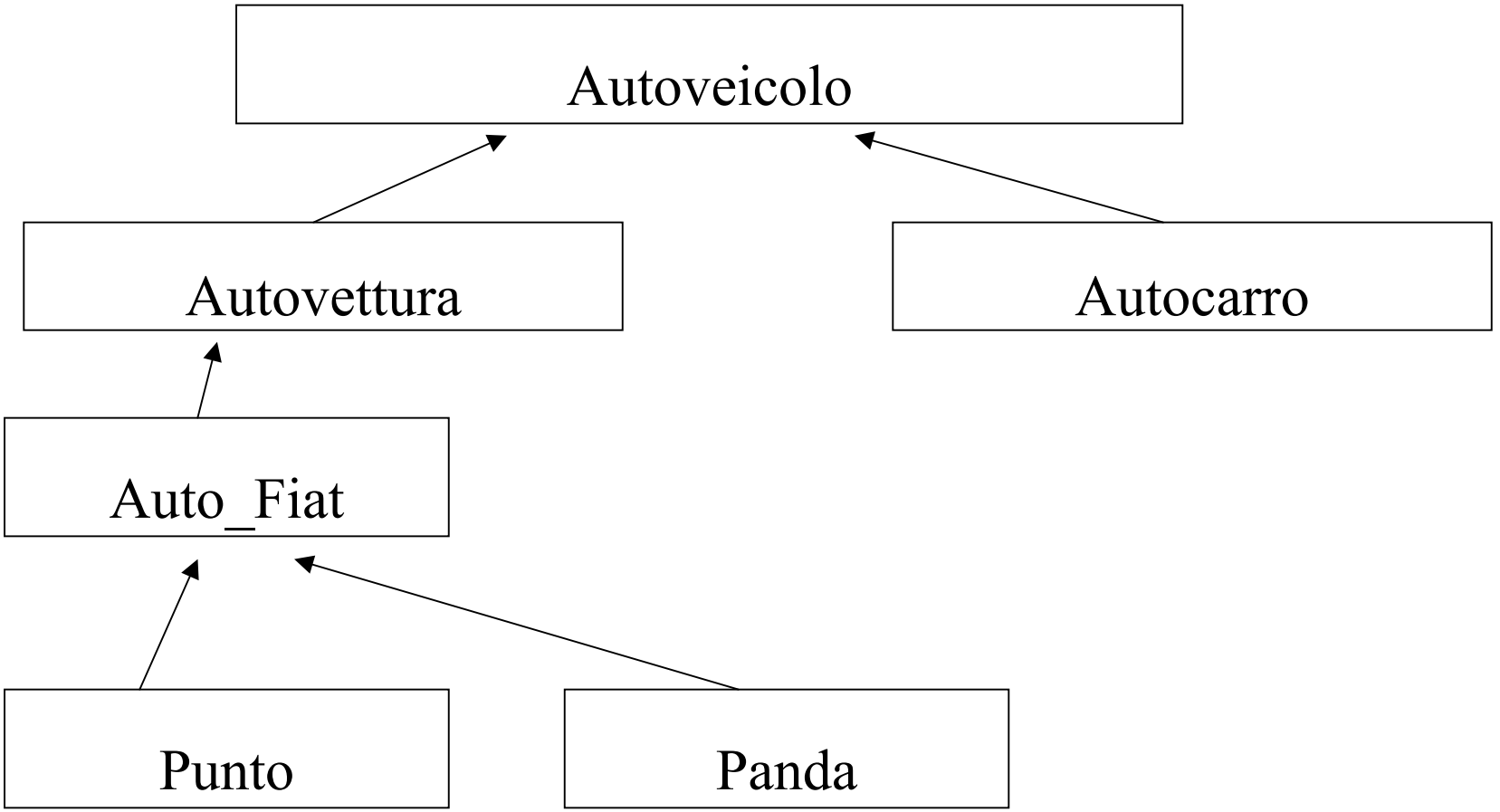
```
#ifndef AUTOVETTURA_H
#define AUTOVETTURA_H

#include "Autoveicolo.h"
#include <iostream>
using namespace std;

class Autovettura : public
Autoveicolo {

 protected:


 public:

  Autovettura() { … }
  virtual ~Autovettura() { … }

  virtual void ChiSei() ;

} ;

#endif
```

I Distruttori Virtuali (4)

```
#include "Autoveicolo.h"
#include "Autovettura.h"
#include <iostream>
using namespace std;

int main() {

  // proviamo con i puntatori
  Autoveicolo* veicolo[2] ;

  cout << endl;
  veicolo[0] = new Autoveicolo() ;
  veicolo[1] = new Autovettura() ;

  cout << endl;
  delete veicolo[0] ;
  delete veicolo[1] ;

return 1;

}
```

```
nbseve(~/polim)>./prv_plom4

 Sto creando un Autoveicolo...
 Sto creando un Autoveicolo...
 Sto creando una Autovettura...

 ...sto distruggendo un Autoveicolo!!!
 ...sto distruggendo una Autovettura!!!
 ...sto distruggendo un Autoveicolo!!!
nbseve(~/polim)>
```

Distruttori Virtuali - Esempio



Creazione di oggetti

Creo un Autoveicolo

Creo una Autovettura

Creo una Auto_Fiat

Creo una Punto

Distruzione di oggetti

(accedendo ad un oggetto tramite puntatore ad Autovettura)

Distruggo una Autovettura

Distruggo un Autoveicolo

Distruzione di oggetti

(accedendo ad un oggetto tramite puntatore + **`virtual`**)

Distruggo una Punto

Distruggo una Auto_Fiat

Distruggo una Autovettura

Distruggo un Autoveicolo

Anche in `CorpoCeleste.h` (e `Sonda.h`) e' opportuno dichiarare `virtual` i Distruttori

Si puo' andare oltre.....?

Posso continuare (verso l'alto) nella catena di ereditarieta'.....

ad esempio da "Autoveicolo" a "mezzi di trasporto su ruote" oppure "mezzi di trasporto"

... sino a che la classe piu' in alto non ha piu' tutti i metodi implementabili!

Classi astratte

Metodi *pure virtual* e classi astratte (1)

1. Se un metodo e' dichiarato `virtual`,
   la classe che eredita (la classe piu' bassa nella catena gerarchica) **puo'** ridefinire il metodo, che e' comunque implementato nella classe da cui si eredita

2. Se un metodo e' dichiarato *pure virtual*
   (`virtual .... = 0 ;`) tutte le classi che ereditano **devono** fornire il metodo, che non e' implementato nella classe da cui si eredita

Metodi *pure virtual* e classi astratte (2)

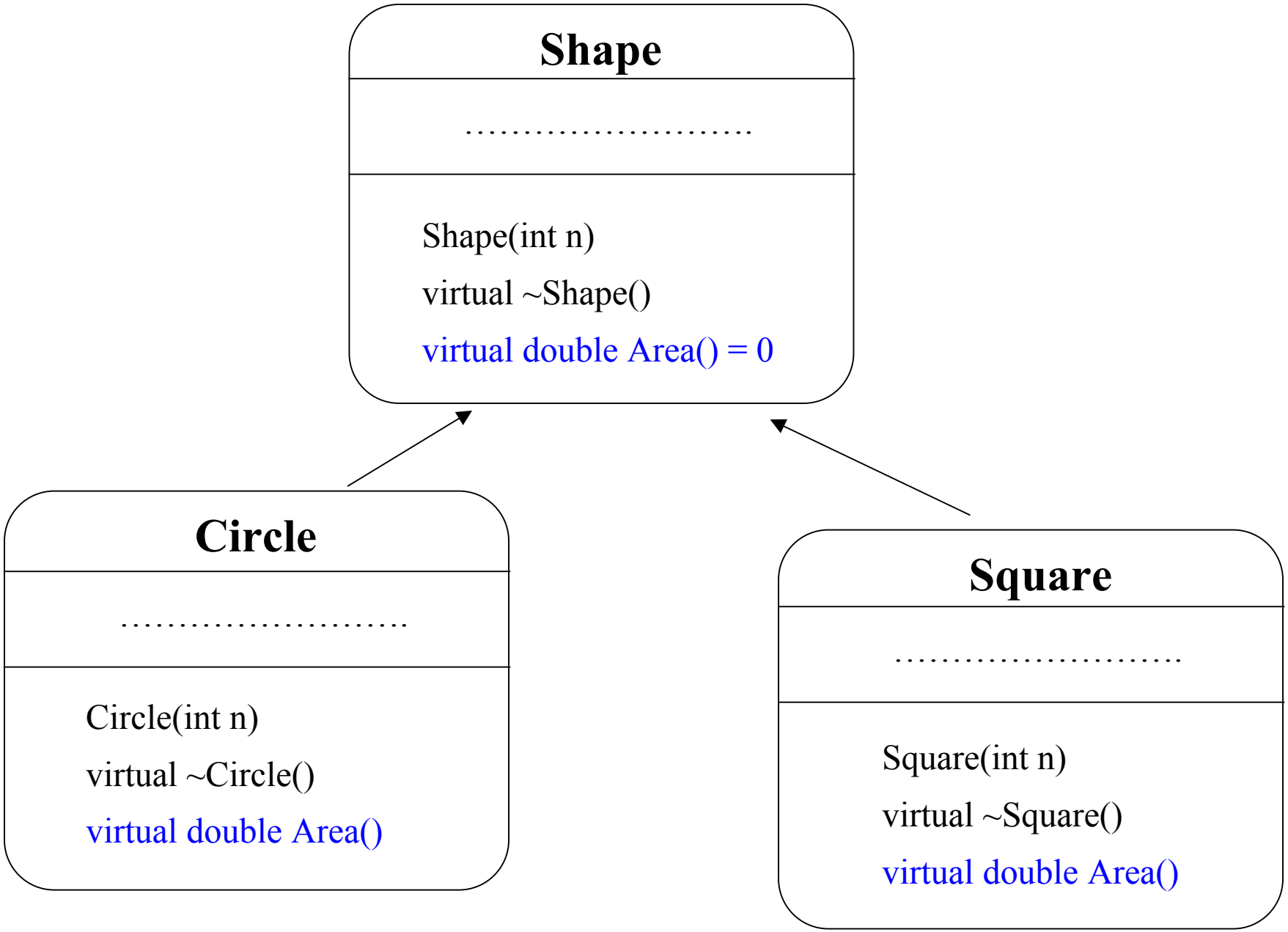
3. Sintassi

```
virtual double Area() = 0 ;
```

4. Una classe con almeno un metodo pure virtual si chiama classe astratta

5. **Attenzione!** Gli oggetti di una classe astratta non possono essere istanziati

6. Esempio: Shape, Circle, Square


Shape
.....
Shape(int n)
virtual ~Shape()
virtual double Area() = 0
Circle
.....
Circle(int n)
virtual ~Circle()
virtual double Area()
Square
.....
Square(int n)
virtual ~Square()
virtual double Area()

18) “per referenza” e “per valore”

L'operatore & (reference) (1)

Consideriamo la dichiarazione con inizializzazione:

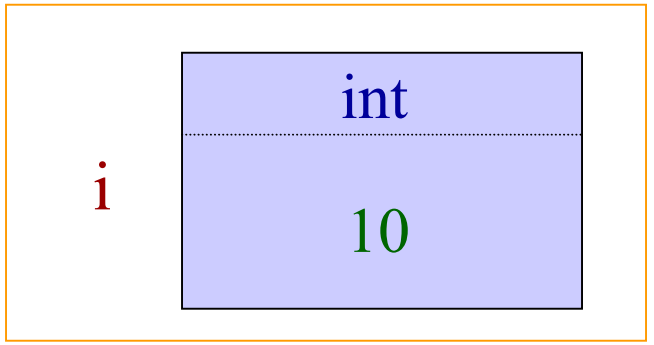
```
int i =10 ;
```

Questa definisce una variabile (istanzia un oggetto) con le seguenti caratteristiche:

identificatore: i

tipo: int

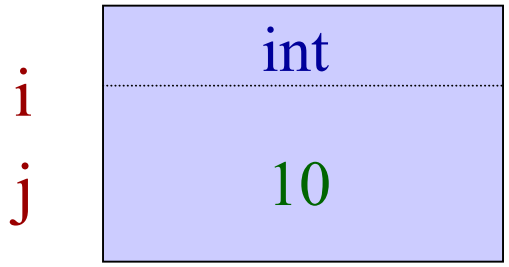
valore: 10



… e lo stesso potrei fare con altri oggetti (float, double, bool, CorpoCeleste…)

Posso collegare alla stessa cella di memoria un altro oggetto dello stesso tipo, tramite l'operatore reference &

```
.............
int i =10 ;
int & j = i ;
.............
```



L'operatore & (reference) (2)

Uso di reference

```
#include <iostream>
using namespace std;

int main() {
    int i  =  5 ;
    int &j =  i ;
    int *q;
    q = new int(i);

    cout << i << " " << j << " " << *q << endl;

    i=8;
    cout << i << " " << j << " " << *q << endl;

return 1;

}
```

```
nbseve(~/prefpval)>op_refer

5 5 5

8 8 5

nbseve(~/prefpval)>
```

L'operatore & (reference) (3)

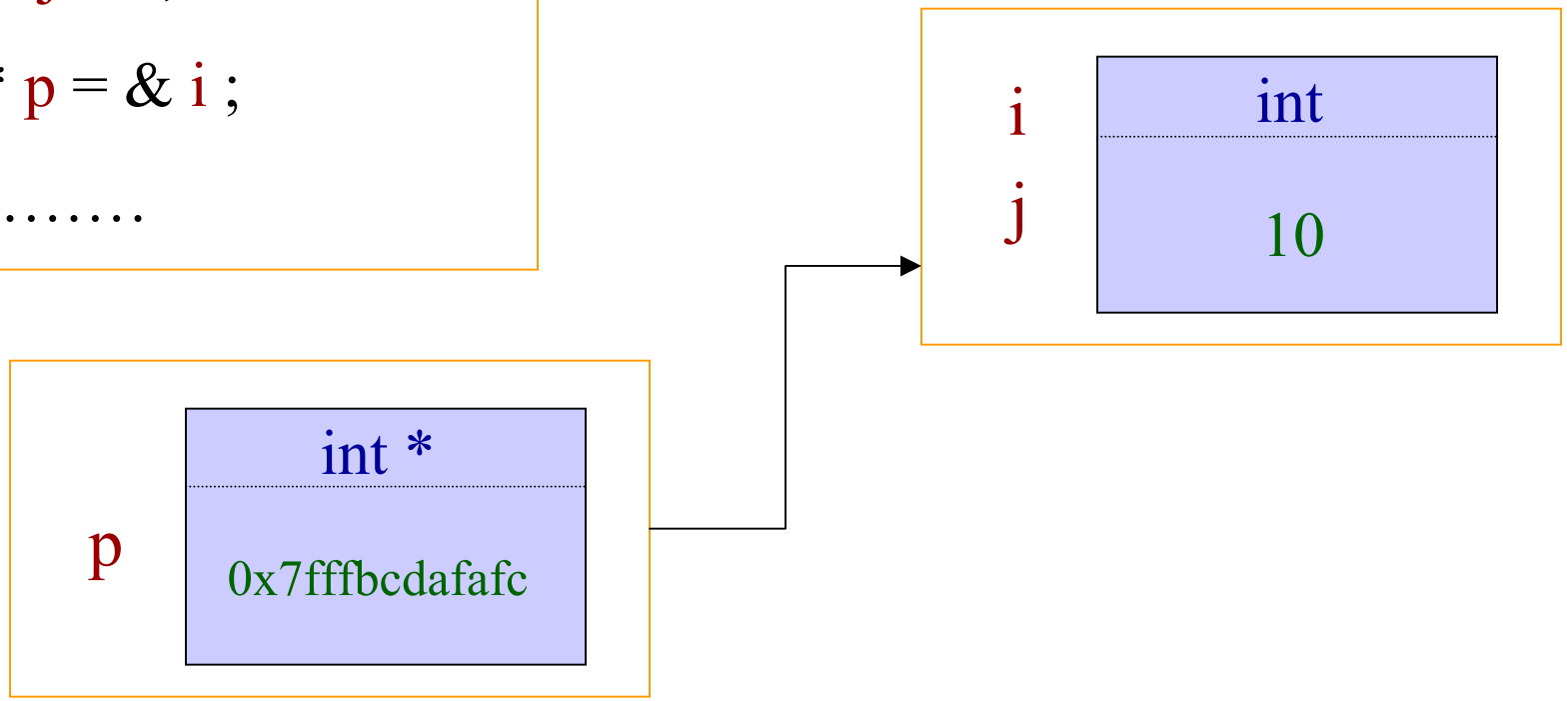
Differenza tra reference e puntatore (1)

```
#include <iostream>
using namespace std;

 int main() {

    int i=5;
    int *p;
    int *q;

    p  = &i;
    q  = new int(i);

    cout << i << " " << *p << " " << *q << endl;

    i=8;
    cout << i << " " << *p << " " << *q << endl;

    return 1;

 }
```

```
nbseve(~/prefpval)>./diff_r_p

5 5 5

8 8 5

nbseve(~/prefpval)>
```

Differenza tra reference e puntatore (2)

```
.............
int i =10  ;
int & j = i ;
int * p = & i ;
.............
```



"per referenza" (*by reference*) .....

Funzione "per referenza" e "per valore"

```
#include <iostream>
using namespace std;

  void swap_ref(int &i1,int &i2) {
     int temp = i1;
     i1 = i2;
     i2 = temp;
  }

  void swap_value(int i1,int i2) {
     int temp = i1;
     i1 = i2;
     i2 = temp;
  }
```

…… e "per valore" (*by value*)

```
#include <iostream>
using namespace std;

int main() {

  int a = 2 ;
  int b = 3 ;

  cout << endl ;
  cout << endl << " a = " << a
               << ";  b = " << b << endl ;

  swap_value(a,b) ;
  cout << endl << " a = " << a
               << ";  b = " << b << endl ;

  swap_ref(a,b) ;
  cout << endl << " a = " << a
               << "; b = " << b << endl ;

  return 1;

}
```

```
a = 2;   b = 3

a = 2;   b = 3

a = 3;   b = 2
```

Nell'esempio precedente le due funzioni sono state chiamate con nomi diversi…

… pero' in genere e' chi scrive la funzione che decide se e' *by reference* o *by value*

```
void swap(int &i1,int &i2);
```

Oppure

```
void swap(int i1,int i2);
```

Nell'esempio precedente e' stata utilizzata una funzione…

… ma esattamente lo stesso accade per i metodi, che possono essere implementati *by reference* o *by value. Ad esempio:*

```
void calcolaPosizione(float & fx, float & fx, float & t) ;
```

Oppure

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

"per referenza" e "per valore" (II)

Funzione "per referenza" e "per valore"

(con *overload* dell'operatore: vedi lezioni successive)

```
#include <iostream>
using namespace std;

 void swap(int *i1,int *i2) {
      int temp = *i1;
      *i1 = *i2;
      *i2 = temp;
 }


 void swap(int i1,int i2) {
      int temp = i1;
      i1 = i2;
      i2 = temp;
 }
```

```
#include <iostream>
using namespace std;

int main() {

  int a = 2;
  int b = 3;

  cout << endl;
  cout << endl << " a = " << a
               << ";  b = " << b << endl ;

  swap(a,b);
  cout << endl << " a = " << a
               << ";  b = " << b << endl ;

  swap(&a,&b);
  cout << endl << " a = " << a
               << ";  b = " << b << endl ;

  return 0;

  }
```

```
a = 2;   b = 3

a = 2;   b = 3

a = 3;   b = 2
```

"per referenza" e "per valore"

Quando usare il passaggio per referenza (&) e quando per valore?

1. Negli argomenti e' preferibile usare il passaggio per referenza, poiche' rende tutto piu' veloce

2. Nella variabile restituita usare il passaggio per referenza (&) con attenzione (vedi lezione su `new`). Nel dubbio non usarlo!

3. Se si vuole evitare che una variabile che e' argomento di un metodo venga modificata, si puo' usare `const` . Ad esempio:

```
void calcolaPosizione(const float & fx,
                      const float & fx, const float & t) ;
```

19) Richiami sulle funzioni

Un esempio di funzione

Noi usiamo (soprattutto) metodi, pero' puo' essere utile saper scrivere una funzione

Un esempio semplice….

```
#include <iostream>
using namespace std

 void swap(int &a, int &b) {
    int c = b ;
    b = a     ;
    a = c     ;
 }


int main() {

   int n1 = 2 ;
   int n2 = 3 ;

   cout << endl << " n1 = " << n1 << "    n2 = " << n2 << endl;
   swap(n1,n2);
   cout << endl << " n1 = " << n1 << "    n2 = " << n2 << endl;

 return 1;

}
```

```
nbacer(~)>g++ -c provafun.cpp
nbacer(~)>g++ provafun.cpp
nbacer(~)>./a.out
 n1 = 2    n2 = 3
 n1 = 3    n2 = 2
```