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)

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)

Premessa: Osservazione sulla Esercitazione di Laboratorio n. 3

Nel corso della esercitazione n. 3 abbiamo utilizzato un oggetto della classe CorpoCeleste per simulare il moto di caduta di un grave

Abbiamo calcolato:

- il tempo di caduta
- la velocita' al momento dell'impatto con il suolo

Queste grandezze sono note in forma analitica:

- il tempo di caduta

$$t = \sqrt{\frac{2h}{g}}$$

- la velocita' al momento dell'impatto con il suolo

$$v = \sqrt{2gh}$$

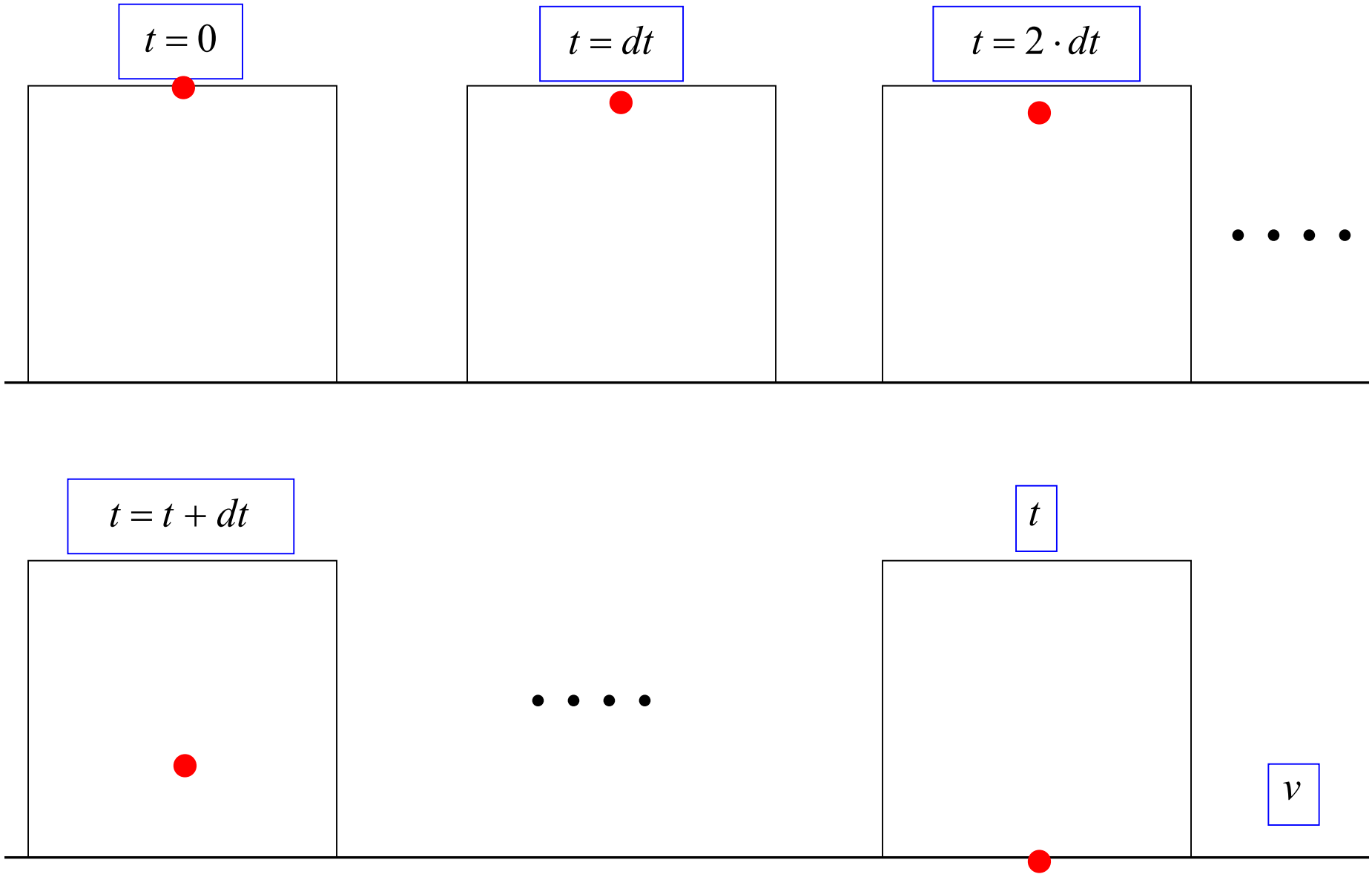
Confrontando il risultato del calcolo numerico con quello dato dalla formula analitica, possiamo avere indicazioni sulla correttezza del programma che abbiamo scritto

Come effettuiamo la simulazione del moto di caduta del grave?

- suddividiamo l'intervallo di tempo di caduta t in una serie di intervallini di tempo dt (dt << t)
- nell'intervallino di tempo dt possiamo utilizzare una descrizione semplificata del moto del grave

utilizziamo una istruzione `while`

- utilizziamo un oggetto della classe `CorpoCeleste`
- utilizziamo il metodo calcolaPosizione (della classe `CorpoCeleste`)


$t = 0$
$t = dt$
$t = 2 \cdot dt$
$t = t + dt$
t
v

7) Puntatori

I Puntatori

Consideriamo la dichiarazione con inizializzazione:

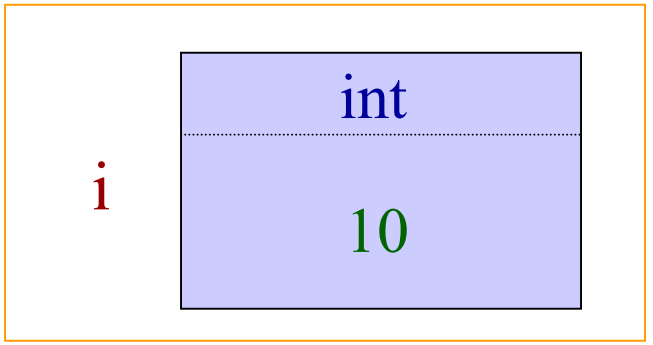
```
int i =10 ;
```

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

identificatore: i

tipo: int

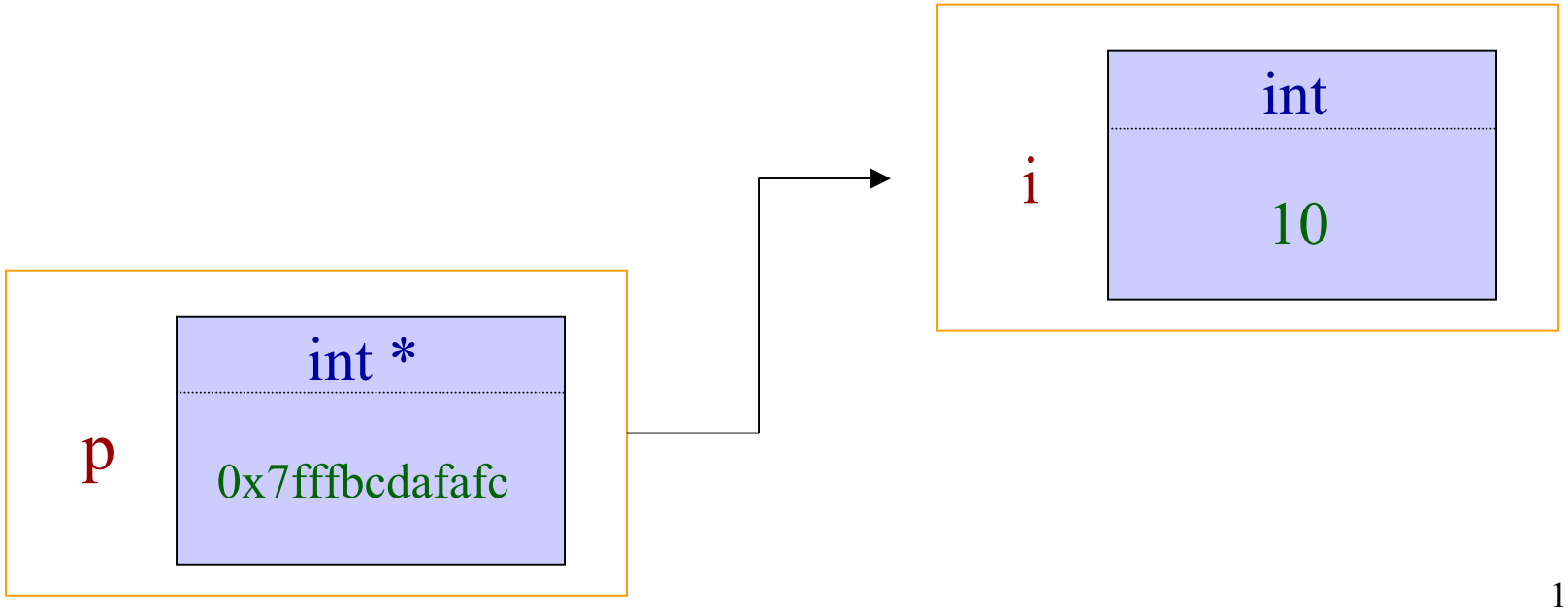
valore: 10



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

Se conoscessimo l'indirizzo di memoria nella quale e' memorizzato l'oggetto (l' int i in questo esempio), potremmo accedere **direttamente all'oggetto stesso**, ed in particolare al suo valore (10 nell'esempio)

Questa operazione e' possibile in C/C++
attraverso l'uso del puntatore !!!



p e' un (altro) oggetto di tipo int* (puntatore ad un intero

Si puo' scrivere (in maniera equivalente)

```
int* p ;

int *p ;
```

Queste due scritture ci ricordano che:

p e' un puntatore ad un intero - di tipo int*

*p e' l'intero a cui p si riferisce - di tipo int

L'oggetto a cui si riferisce il puntatore si ottiene (*deferencing*) tramite l'asterisco *. Nell'esempio:

utilizzare `*p` e' come utilizzare `i`

Lo stesso si puo' fare con tutti gli (altri) oggetti!!!

Ad esempio:

```
double * q ;          // puntatore ad un double  (tipo double*)

CorpoCeleste * c ;    // puntatore ad un CorpoCeleste
                                  (tipo CorpoCeleste*)

bool * b ;            // puntatore ad un oggetto booleano (tipo bool*)
```

Per ottenere l'indirizzo di un oggetto, si usa l'operatore & (*reference*). Nell'esempio:

```
int  i   ;

int * p = & i  ;
```

I metodi di un oggetto si applicano al corrispondente puntatore tramite la freccetta (digitare "meno" e poi "maggiore" ->). Ad esempio:

```
CorpoCeleste  Marte("marte", 1., 1., 2., 3., 4.) ;

CorpoCeleste * pM = & Marte  ;

pM -> calcolaPosizione();
```

Attenzione alle ambiguita' nella scrittura:

```
CorpoCeleste  Marte("marte", 1., 1., 2., 3., 4.) ;

CorpoCeleste * pM = & Marte  ;

 pM -> calcolaPosizione();          //OK

pM ->X();                           //OK

(*pM).X() ;                         //OK. Equivale al precedente

*pM  . X();                         //NO. Ambiguo!
                                    //Potrebbe essere anche *(pM  . X()) e il
                                      compilatore non sa scegliere;
```

Un esempio

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

int main() {

        int i  = 10  ;
        int *p = & i ;

        cout << endl ;

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

        cout << endl ;


        return 1;

}
```

```
nbseve(~/LabC_L5)>./provap1

 i = 10
 p = 0x7fff2ef5df44
*p = 10
```

Un altro esempio

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

int main() {

        int i  = 5  ;
        int j  ;
        int *p ;

        p = &i ;
        j = i  ;

        cout << i << *p << j << endl ;

        i = 8 ;
        cout << i << *p << j << endl ;

        return 1;
}
```

```
nbseve(~/LabC_L5)>./provap2
5 5 5
8 8 5
```

Vantaggi nell'uso dei puntatori (2)

Vantaggi per il *C/C++*:

- velocizzano l'accesso ai dati in memoria
  (ad esempio la gestione di matrici di grandi dimensioni)
- permettono di allocare e liberare memoria durante la fase di esecuzione del programma (vedi prossima lezione)
- Permettono di condividere informazioni tra diversi blocchi di un programma senza duplicare in memoria strutture dati di grandi dimensioni.

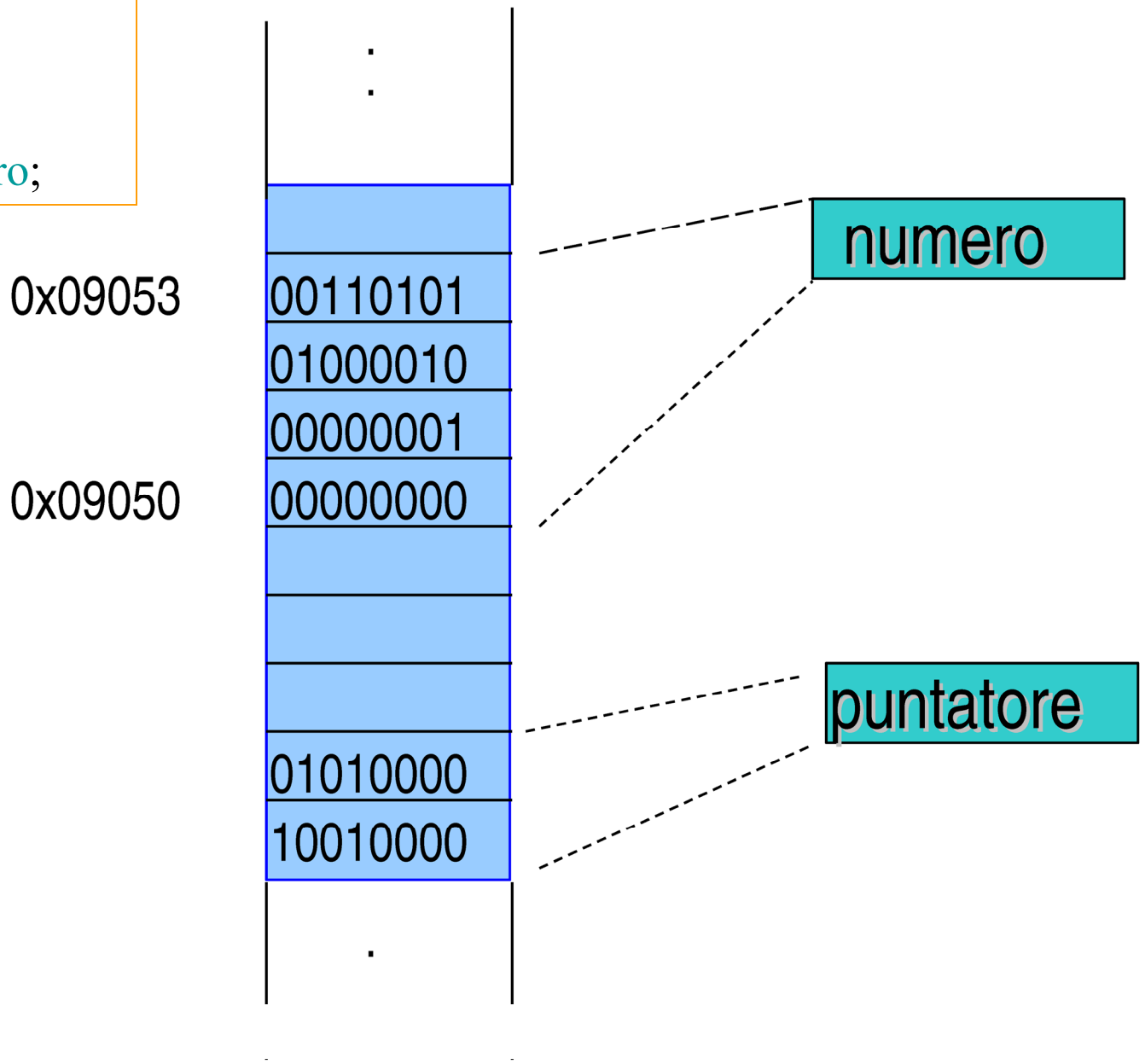
Vantaggi nell'uso dei puntatori (1)

Vantaggi specifici del C++

Straordinariamente potente se unito ad Ereditarieta' e Polimorfismo (dovete attendere qualche lezione!)

Rende molto flessibile la gestione di oggetti diversi, ma tra loro simili (ad esempio Sonda e CorpoCeleste, oppure Cerchio, Quadrato e Triangolo, ecc.)

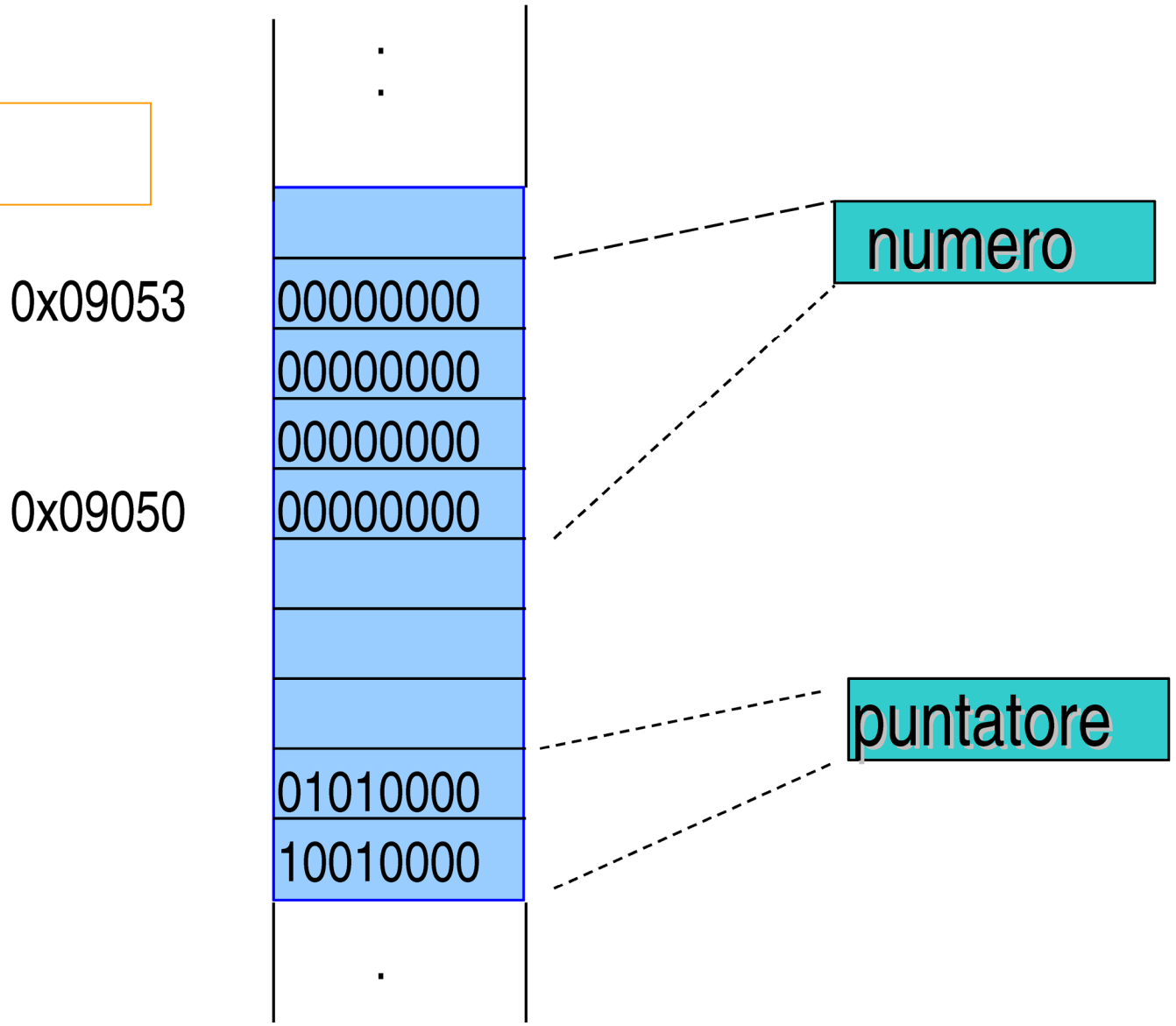
```
int numero ;

numero = 82485;

int *puntatore;

puntatore = &numero;
```



```
numero = 0 ;
```

oppure

```
*puntatore = 0 ;
```



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

int main() {

        int numero  = 82485 ;
        int *puntatore = & numero    ;

        cout << " numero     = " << numero
             << " puntatore = " << puntatore << endl ;

        numero = 0 ;
        cout << " numero     = " << numero
         << "      puntatore = " << puntatore << endl ;

        cout << endl ;
        return 1;
}
```

```
nbseve(~/LabC_L5)>./provap3
numero    = 82485  puntatore = 0x7fff9dbc6ed4
numero    = 0      puntatore = 0x7fff9dbc6ed4
```

7) Puntatori

8) Vettori (Array)

Vettori/Array

float	float		float
10.	12.9	....	-8.4

- Un vettore (array) è una sequenza di variabili tutte dello stesso tipo che occupano locazioni di memoria contigue.
- Dichiarazione di un vettore di oggetti del tipo Tipo:

  Tipo identificatore[dimensione];
- Uso di un elemento del vettore:

  identificatore[elemento]
- Dove dimensione e elemento sono degli oggetti di tipo int

NB

- Contrariamente ad altri linguaggi in C e C++ il primo elemento di un vettore ha indice 0 e l'ultimo, se n è la dimensione del vettore, ha indice n-1.

- Come in tutti i linguaggi la dichiarazione del vettore ed il suo dimensionamento comportano l'allocazione della memoria necessaria per la dimensione dichiarata: se si prova ad usare un elemento con indice superiore a n-1 si ha uno **sfondamento della memoria**, errore che comporta problemi in punti imprevedibili del programma!

Esempi di dichiarazioni

```
int V[25] ;        // dichiarazione di un vettore V
                   // con 25 elementi integer:  V[0], V[1]… V[24]

double vdb[3] ;    // dichiarazione di un vettore
                   // vdb con 3 elementi double:
                   // vdb[0], vdb[1], vdb[2]

string word[50] ;  //dichiarazione di un vettore word
                   //con 50 elementi string: word[0],…,word[49]
```

Inizializzazione

```
int V[25] = {3,   5,   6,   1} ;

double vdb[3] = {-1.2,   4.7,   5.9} ;

string student[4] = {"Chris Berkley", "Kevin Chao",
                                    "Missy Mesfin", "Joel Triemstra"};

int M[ ] = {1, 4, 15, 2};

string Name[ ] = "Michael Bird" ;
```

Esempio di inizializzazione e uso

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

int main() {

      string studenti[4] ;

      studenti[0] = "Chris Berkley"  ;
      studenti[1] = "Kevin Chao"     ;
      studenti[2] = "Missy Mesfin"   ;
      studenti[3] = "Joel Triemstra" ;

      cout <<
       " Elenco studenti del corso di Laboratorio di Calcolo (A)"
           << endl ;

      for(int i=0 ; i<4 ; i++ ) {

          cout << " "  << i+1 << " " << studenti[i] << endl ;
      }

return 1;
}
```

```
nbseve(~/LabC_L5)>./provlstu
 Elenco studenti del corso di Laboratorio di Calcolo (A)
 1 Chris Berkley
 2 Kevin Chao
 3 Missy Mesfin
 4 Joel Triemstra
```

Vettori/Array multidimensionali (matrici)

- Dichiarazione di una matrice con rDim righe e cDim colonne

```
tipo  identificatore[rDim][cDim] ;
```

- Esempio:

```
double  tabella[3] [4] ;
```

Dichiarazione ed inizializzazione

Per l'inizializzazione si elencano gli elementi della prima riga, poi della seconda etc…

```
int  M[3][4] = {{1,2,3,4},
                {5,6,7,8},
                {9,10,11,12}} ;
```

oppure

```
int  M[3][4] = {1,2,3,4,5,6,7,8,9,10,11,12} ;
```

Esempio: prodotto scalare

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

 int main( ) {
     double A[3],B[3];
     double pScal=0.;

     cout  <<"inserisci gli elementi del vettore A"<< endl;
     cin >> A[0] >> A[1] >> A[2];

     cout  <<"inserisci gli elementi del vettore B"<< endl;
     cin >> B[0] >> B[1] >> B[2];

     for(int j=0; j<3; j++) {
             pScal += A[j]*B[j] ;
     }
     cout << " A scalare B vale " << pScal << endl;

 return 1;
}
```

Esempio: prodotto di matrici

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

int main() {

  int M1[3][2] = {  {1,0},   {2,1},   {0,2} };
  int M2[2][3] = {  {0,1,2}, {1,2,1} } ;
  int M3[3][3] ;

  for(int j=0; j<3; j++) {          //loop sulle righe di M3
     for(int k=0; k<3; k++) {       //loop sulle colonne di M3

        M3[j][k] = 0 ;
        for(int l=0; l<2; l++) {  //loop su colonne M1 e righe M2

           M3[j][k] += M1[j][l]*M2[l][k] ;

        }
     }
  }

  return 1;

}
```

Osservazione C++!

Ma non si potrebbe definire una classe di tipo Matrix (almeno per le matrici quadrate) e su di essa implementare una operazione di moltiplicazione (o un metodo…)

E poi scrivere…

```
Matrix M1( … ) ;
Matrix M2(….)  ;
Matrix M3 = M1*M2  ;
```

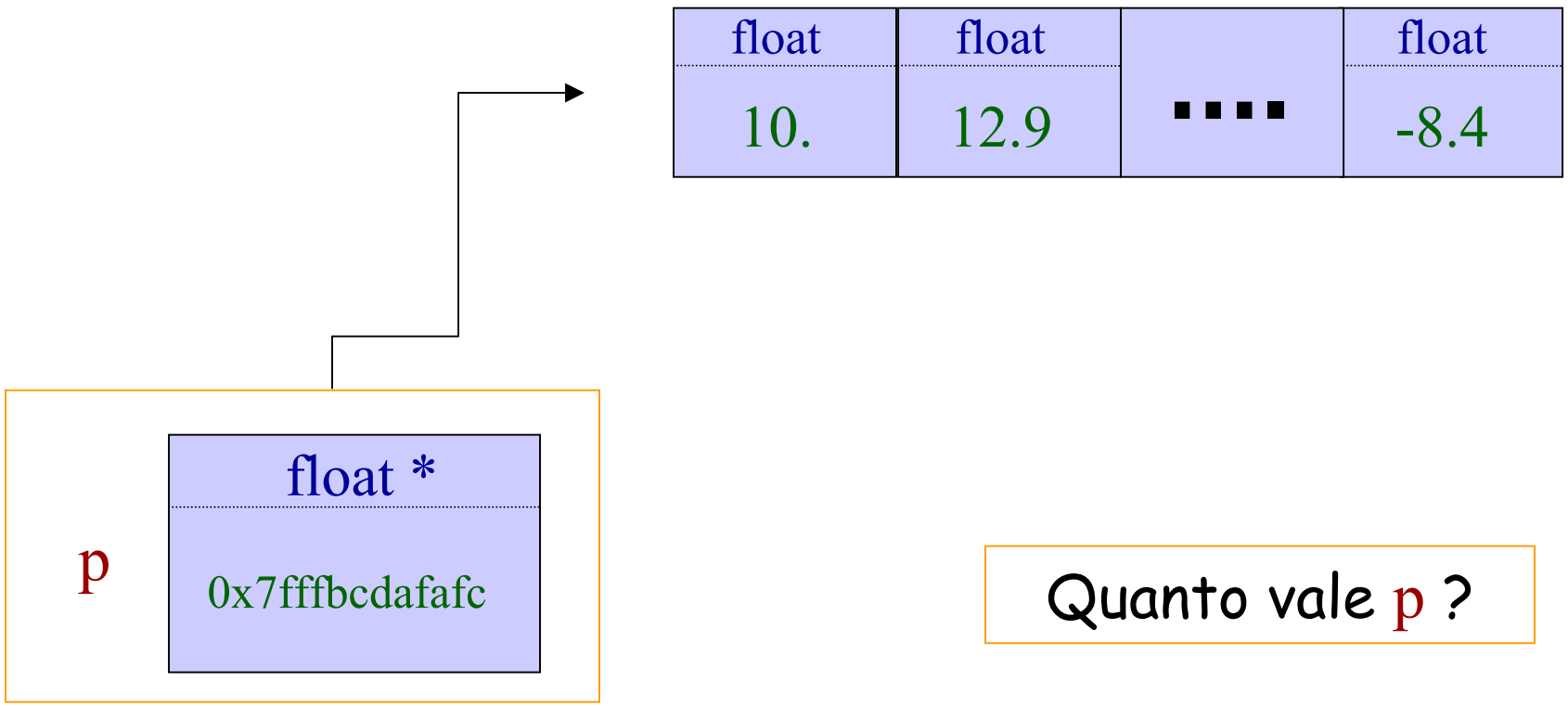
???

7) Puntatori

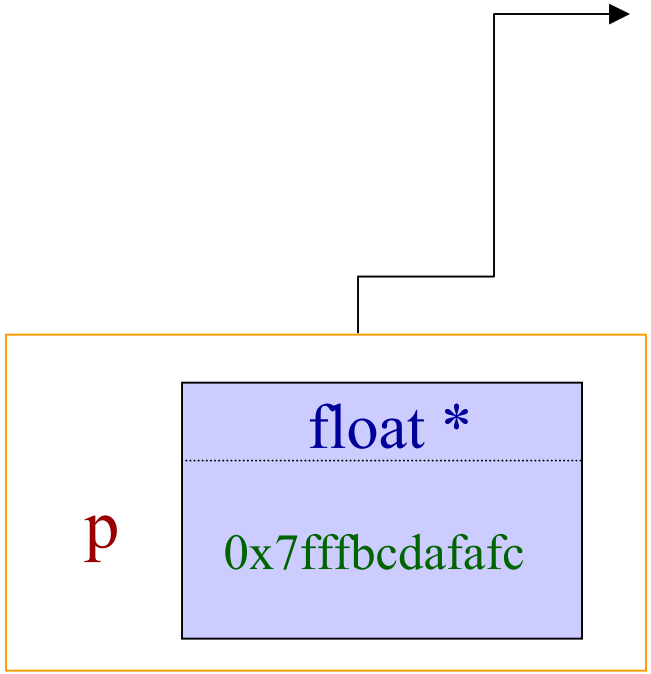
8) Vettori (Array)

9) Vettori e Puntatori

Puntatore ad un vettore



Quanto vale p ?



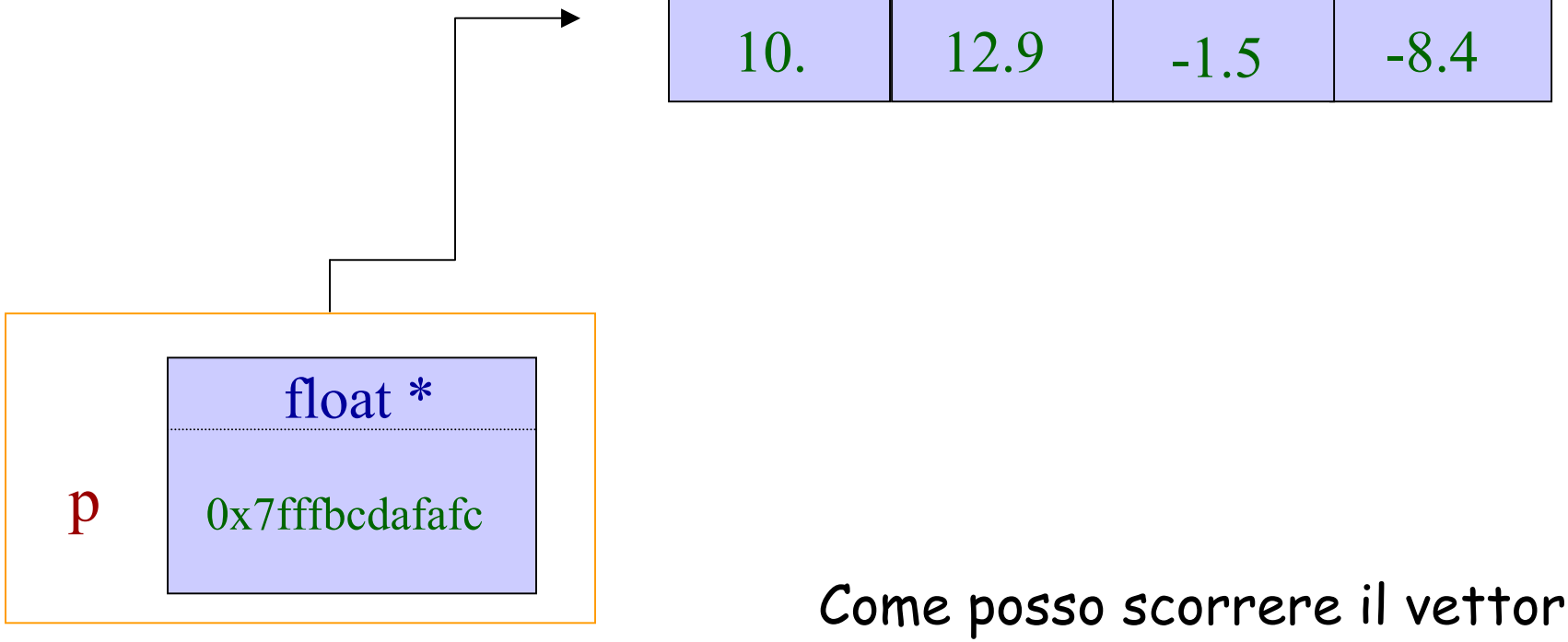
v[0]	v[1]	v[2]	v[3]
float	float	float	float
10.	12.9	-1.5	-8.4

```
float v[4] ;
float * p  ;
p = & v[0] ;
```

Oppure, piu' semplicemente

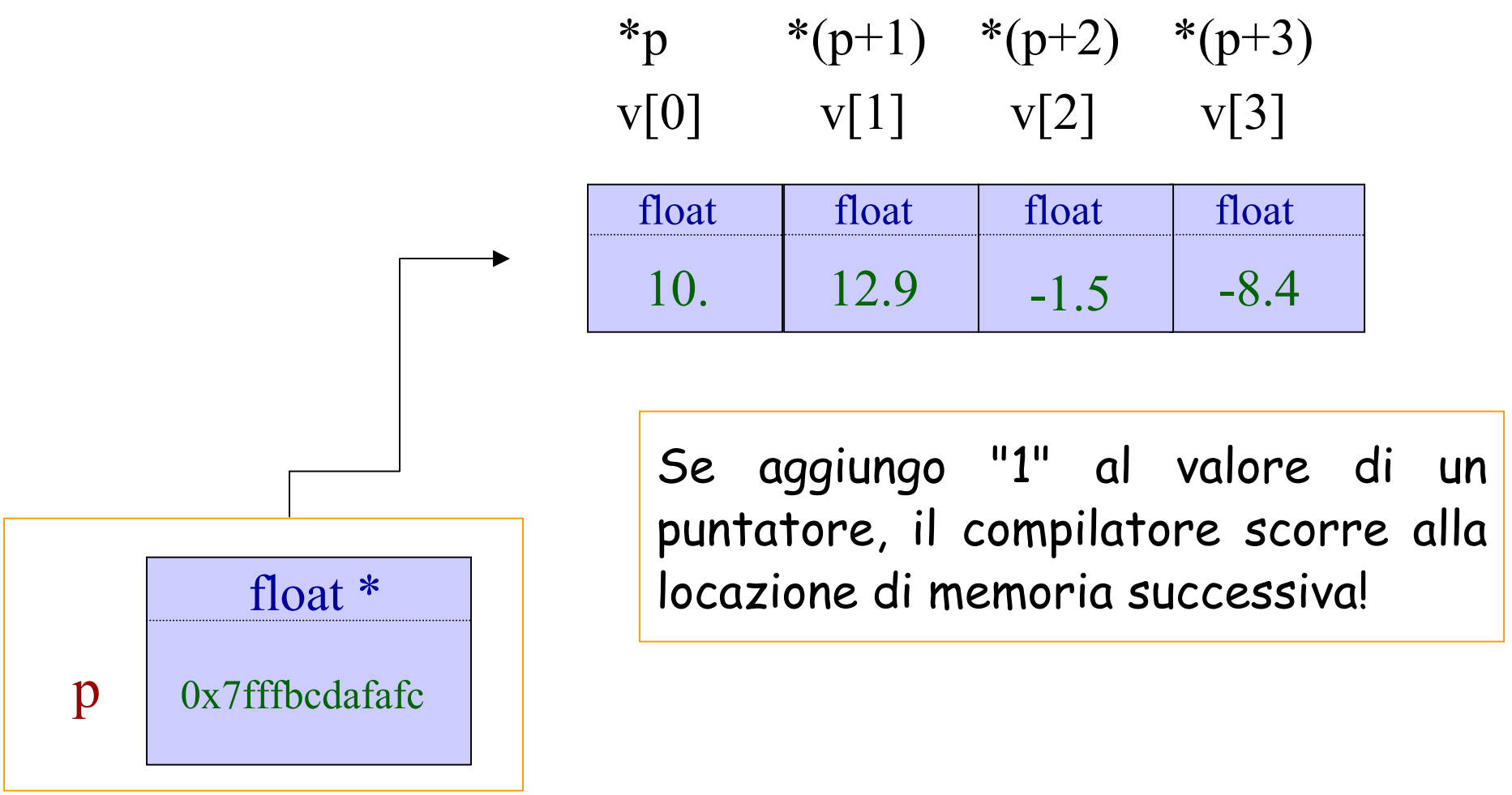
```
float v[4] ;
float * p  ;
p  =  v    ;
```

```
float v[4] ;
float * p  ;
p  =  v    ;
```



Come posso scorrere il vettore utilizzando i puntatori?

Aritmetica dei Puntatori



Se aggiungo "1" al valore di un puntatore, il compilatore scorre alla locazione di memoria successiva!

Esempi (1)

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

int main() {

      string studenti[4] ;

      studenti[0] = "Chris Berkley"  ;
      studenti[1] = "Kevin Chao"     ;
      studenti[2] = "Missy Mesfin"   ;
      studenti[3] = "Joel Triemstra" ;

      cout <<
       " Elenco studenti del corso di Laboratorio di Calcolo (A)"
          << endl ;

      string * p = studenti;

      for(int i=0 ; i<4 ; i++ ) {

          cout << " "  << i+1 << " " << *(p++) << endl ;
      }

return 1;
}
```

```
nbseve(~/LabC_L5)>./provlstp
 Elenco studenti del corso di Laboratorio di Calcolo (A)
 1 Chris Berkley
 2 Kevin Chao
 3 Missy Mesfin
 4 Joel Triemstra
```

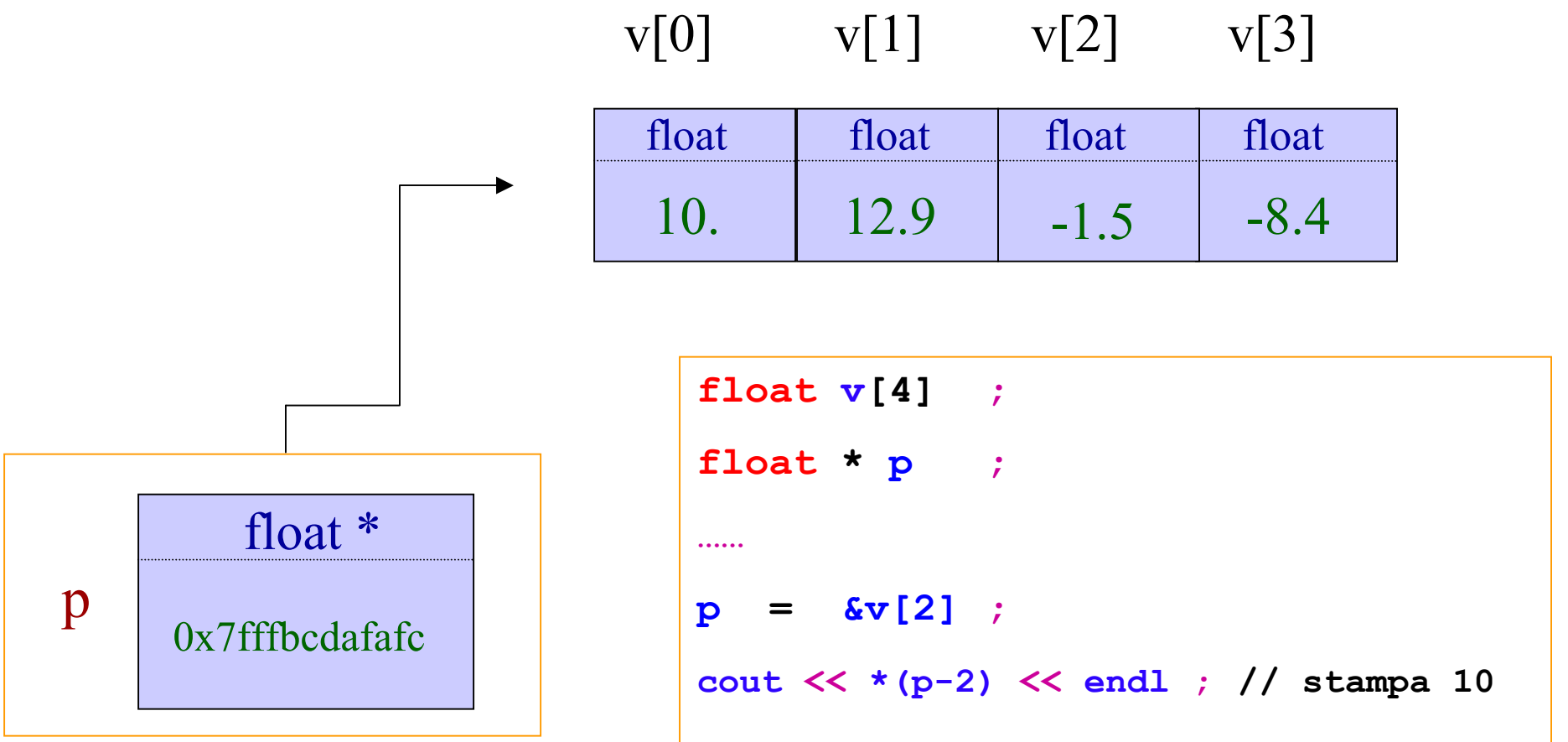
Avreste anche potuto scrivere:

```
cout << " "  << i+1 << " " << *(studenti+i) << endl ;
```

Pero' `studenti` non e' un puntatore, ma la label del vettore!

Si comporta (in parte) come un puntatore

Posso anche sottrarre ad un puntatore un numero intero per scorrere il vettore all'indietro:




```
float v[4]   ;
float * p    ;
.....
p   =   &v[2] ;
cout << *(p-2) << endl ; // stampa 10
```

Esempi (2)

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

int main() {

    int i[3] ;
    int * p = i ;

    double f[3] ;
    double * q = f ;

    i[0] = 0  ;
    i[1] = 1  ;
    i[2] = 2  ;

    f[0] = 10.  ;
    f[1] = 11.  ;
    f[2] = 12.  ;

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

    cout <<         p+1    << "  " << q+1    << "   "
                 << *(p+1) << "  " << *(q+1) << endl ;

    cout <<         p+2    << "  " << q+2    << "   "
                <<  *(p+2) << "  " << *(q+2) << endl << endl ;

return 1;
}
```

```
nbseve(~/LabC_L5)>./provpA

0x7fffa645fe80  0x7fffa645fe60   0  10
0x7fffa645fe84  0x7fffa645fe68   1  11
0x7fffa645fe88  0x7fffa645fe70   2  12
```

I puntatori sono numeri interi, pero':

p e' di tipo int*

q e' di tipo double *

Cambia l'algebra!

Se proviamo a stampare i valori dei puntatori p , p+1, p+2 troviamo due indirizzi che distano tra loro 4 byte.

Se proviamo a stampare i valori dei puntatori q , q+1, q+2 troviamo due indirizzi che distano tra loro 8 byte.

In un vettore di int gli indirizzi distano 4 byte, in un vettore di double distano 8 byte

7) Puntatori

8) Vettori (Array)

9) Vettori e Puntatori

10) Classe SistemaSolare (prima parte)

10) Classe SistemaSolare (prima parte)

La Classe SistemaSolare: Attributi e Metodi

SistemaSolare

????????? pianeti
int N

SistemaSolare(int n)

~SistemaSolare()

int aggiungiPianeta(CorpoCeleste * unPianeta)

void evolvi(float T, float dt)

int nPianeti()

SistemaSolare.h

SistemaSolare.cc

Ricordando la Struttura ……

e l'*Interaction-diagram* della simulazione

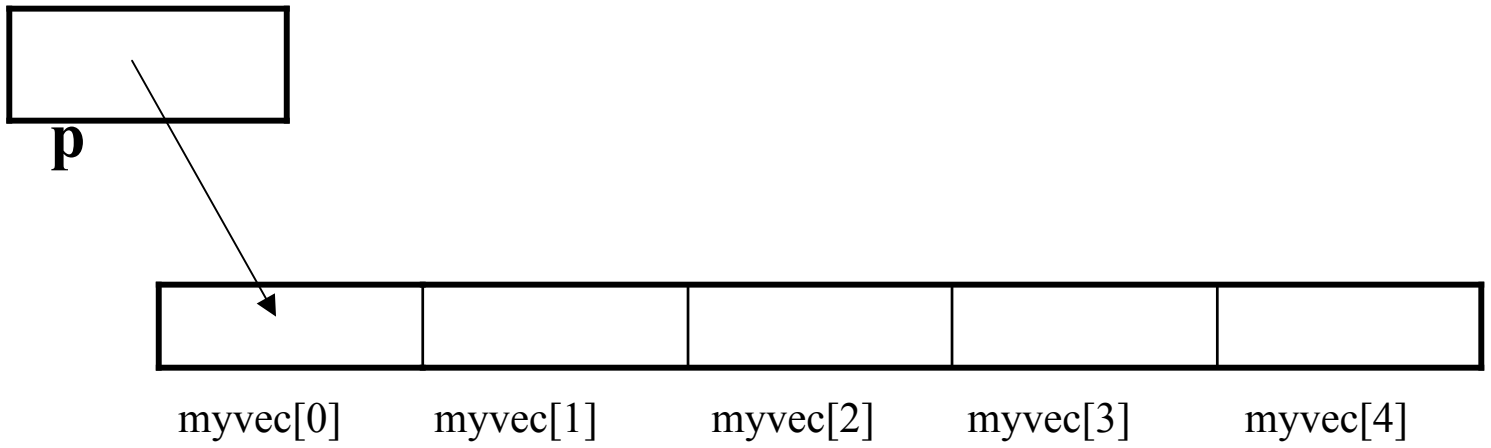
SistemaSolare.h

```
#include "CorpoCeleste.h"
#define G 6.673e-11

class SistemaSolare {
  protected:
    ??????????  pianeti;
                 // lista dei pianeti
    int N;       // numero dei pianeti

  public:
    SistemaSolare(int n);
    ~SistemaSolare() ;
    int aggiungiPianeta(CorpoCeleste *unPianeta);
    int nPianeti() {return N;} ;
    void evolvi(float T, float dt) ;

};
```

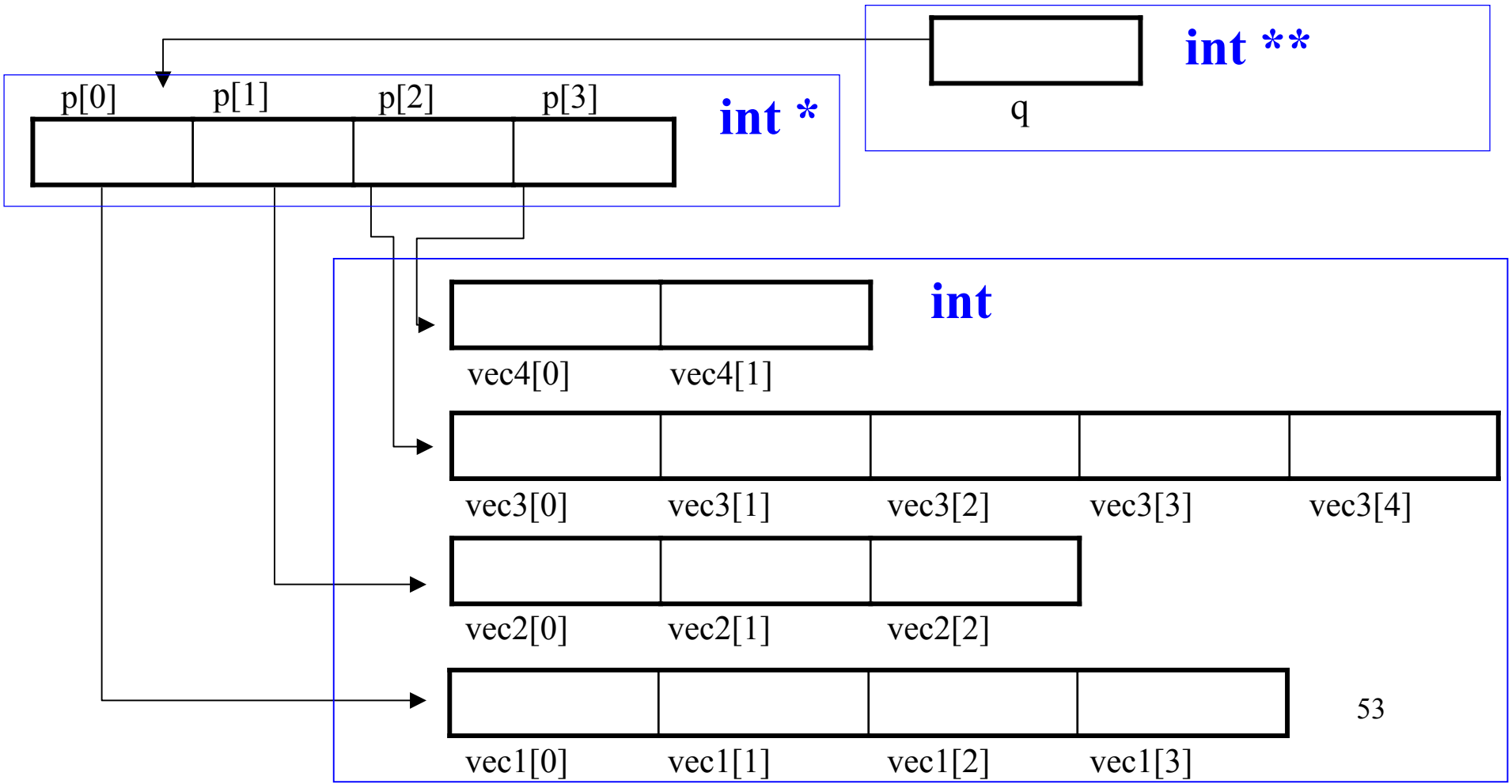
La Struttura Dati di Sistema Solare (I)

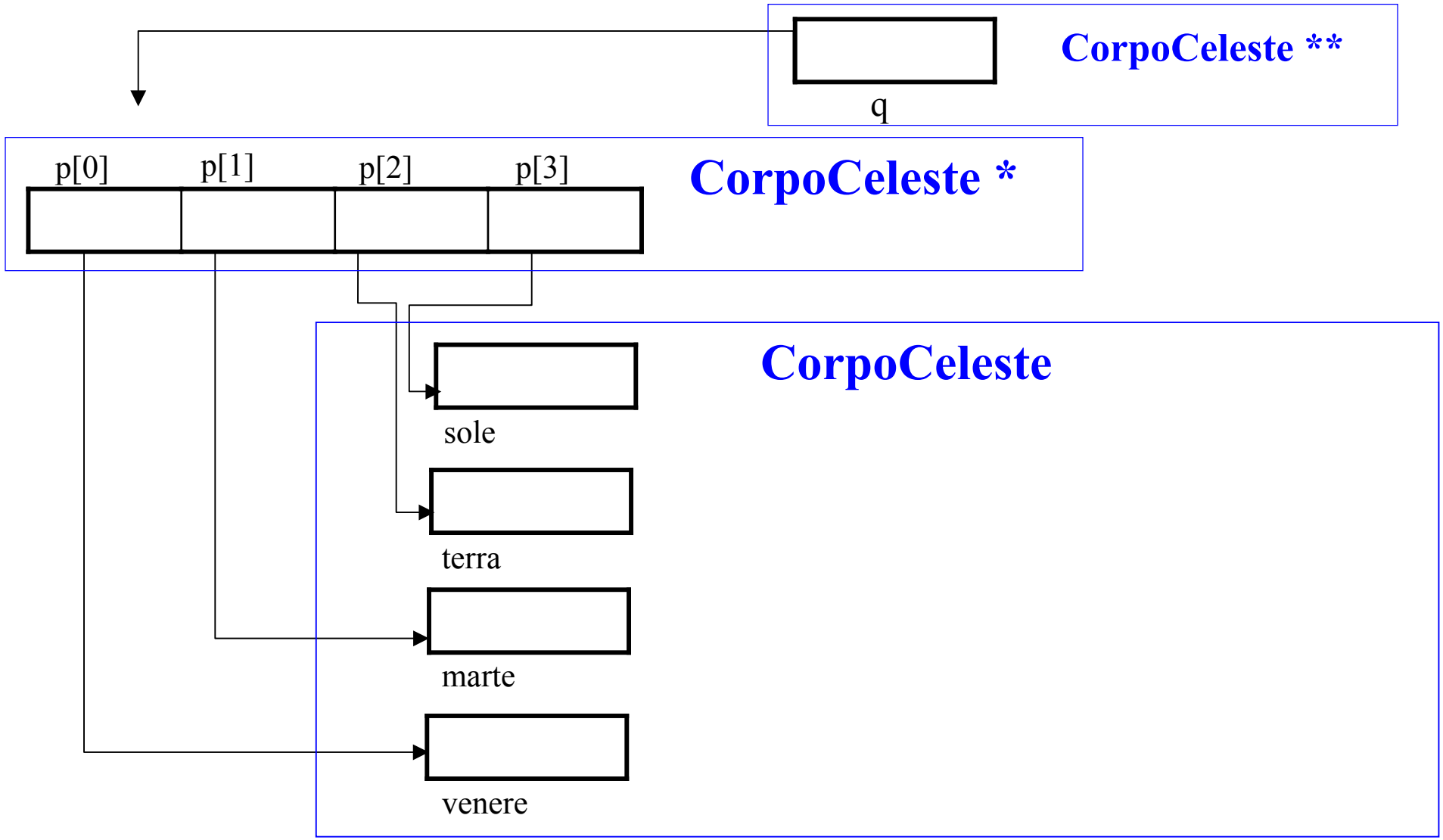
Puntatori e Vettori

```
int myvec[5];

int * p;

p =  myvec;
```



```
int vec1[4],vec2[3],vec3[5],vec4[2];
int * p[4];
p[0] =  vec1; p[1] =  vec2;
p[2] =  vec3; p[3] =  vec4;
int ** q;   q = p;
```




CorpoCeleste **
q
p[0]
p[1]
p[2]
p[3]
CorpoCeleste *
CorpoCeleste
sole
terra
marte
venere

SistemaSolare.h

```
#ifndef SISTEMASOLARE_H
#define SISTEMASOLARE_H

#include "CorpoCeleste.h"
#define G 6.673e-11

class SistemaSolare {
  protected:
    CorpoCeleste ** pianeti;      // lista dei pianeti
    int N;                        // numero dei pianeti

  public:
    SistemaSolare(int n);
    ~SistemaSolare();
    int aggiungiPianeta CorpoCeleste *unPianeta);
    int nPianeti() {return N;};
    void evolvi(float T, float dt);
};

#endif
```

SistemaSolare.cc (1)

```
#include "SistemaSolare.h"
#include <cstdlib>
#include <cmath>
#include <iostream>

SistemaSolare::SistemaSolare(int n) {
    N  = 0;        // si pone il numero iniziale di
                   // pianeti pari a 0
    ??????????   // non sappiamo come costruire il
                   // contenitore per i pianeti!!!

}
 ..........................................
```

Affronteremo questo problema nella prossima lezione!