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)

Premessa: Osservazione sulla Esercitazione di Laboratorio n. 4

Nel corso della esercitazione n. 4 abbiamo utilizzato un oggetto della classe CorpoCeleste per simulare il moto di caduta di un grave in un fluido viscoso

La differenza principale tra la situazione dell'esercitazione 3 e quella dell'esercitazione 4 consiste nel fatto che la forza a cui e' soggetto il corpo dipende dalla velocita' del corpo nel fluido...

...e quindi deve essere calcolata all'interno del ciclo `while`, utilizzando il valore "istantaneo" della velocita'

Premessa:
Osservazione sulle Esercitazioni di Laboratorio n. 3 e n. 4

Nel corso della esercitazione n. 3 abbiamo utilizzato un oggetto della classe CorpoCeleste per calcolare una quantita' (ad esempio la velcoita' finale o il tempo di caduta) il cui valore e' determinabile per via analitica.

In questo modo abbiamo controllato il buon funzionamento della Classe CorpoCeleste (*debug*)

Nell'esercitazione n. 4 abbiamo utilizzato un oggetto della classe CorpoCeleste per calcolare una quantita' (ad esempio la velcita' finale o il tempo di caduta) il cui valore e' (molto) piu' difficile da determinare per via analitica.

11) Gestione dinamica della memoria

"Indice"

Come viene gestita la memoria in C++ ?

1. Che cosa vogliamo ? il problema
   - Gestione dinamica della Memoria
   - Allocazione della memoria
     (*compilation time* e *run time*)
   - *Out of scope*
   - Puntatori ed Oggetti puntati

2. Come risolverlo ? la soluzione

Premessa: Osservazione sulla Esercitazione di Laboratorio

Nel corso delle esercitazioni in laboratorio verificherete che se create un oggetto all'interno di uno *scope*, ne memorizzate il puntatore in un vettore, al di fuori dello *scope* l'oggetto viene distrutto ed il risultato e' imprevedibile

ad esempio…

Programma **ERRATO** con l'uso di un vettore di puntatori ad oggetti
(molto piu' elegante … **se** fosse giusto)

==========

1

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

int main () {

// uso un vettore di puntatori ad oggetti
// di tipo Corpoceleste

    const int np = 6;
    CorpoCeleste * Pianeti[np];

// Con un loop scrivo gli elementi del vettore

    for (int i=0; i<np; i++) {
      CorpoCeleste Ausiliario("",i,i,i,i,i);
      Pianeti[i] = &Ausiliario;
    }
```

```
// Con un loop posso estrarre le informazioni.
// Ad esempio la massa e la posizione

 cout << endl
   << "Pianeta n.      Massa        Posizione "  << endl
   << "                             x       y   " << endl << endl;

 for ( int i=0; i<np; i++ ) {
   cout  << "     " << i+1 <<  "              "
         << Pianeti[i]->M() << "         "
         << Pianeti[i]->X() << "    "  << Pianeti[i]->Y() << endl;
 }

 cout << endl ;

// Errato: ottengo qualcosa che puo' essere
//             -- errore (a run time o compilation time)
//             -- numeri a caso
//             -- valori dell'ultimo pianeta


  return 1;
}
```

Il punto della situazione: dove siamo arrivati?

Classe SistemaSolare

Abbiamo scritto il puntatore che ci permette di accedere alla lista dei (puntatori ai) pianeti nella forma

```
CorpoCeleste ** pianeti ;
```

Non sappiamo come scrivere il contenitore nel quale memorizzare la lista dei (puntatori ai) pianeti

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!!!

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

Che cosa vogliamo ? il problema (1)

Gestione dinamica della Memoria

1. Rilevanza per un Programmatore Scientifico
   → Utilizziamo dati ←

2. Un esempio da una ipotetica classe CorsoLabCalc
   - Variante a dimensione **fissa**

```
class CorsoLabCalc {
  protected:
      string NomiStud[20] ;
      ..............
 } ;
```

   - Variante a dimensione **variabile** (???)

```
class CorsoLabCalc {
  protected:
      int n;
      string NomiStud[n]
      ..............
 } ;
```

Che cosa vogliamo ? il problema (2)

3. Osservazioni sulle proposte precedenti
   - Variante a dimensione **fissa**
     - Riserva sempre lo spazio per i nomi di 20 studenti
     - Enorme spreco di memoria se gli studenti sono molto pochi
     - Spazio insufficiente se gli studenti sono piu'di 20
     - Come si libera spazio (come posso cancellare la lista degli studenti dell'anno 2020-21 quando inizia l'anno 2021-22)?
   - Variante a dimensione **variabile** (???)
     - Non funziona (**errata**)
     - E' quello che vorremmo fare (se funzionasse) per avere flessibilita' nel numero di studenti iscritti
     - Non risolverebbe (anche se funzionasse) il problema della riassegnazione della memoria

Che cosa vogliamo ? il problema (3)

4. Applicazione a casi di fisica
   - In fisica si acquisiscono dati, ad esempio da una serie successiva di eventi dello stesso tipo
   - Dello "stesso tipo" significa che vengono gestiti dallo stesso software ma non che hanno tutti le stesse dimensioni (fluttuazioni statistiche)
   - Un evento deve essere cancellato quando si passa all'esame dell'evento successivo

Proviamo ad affrontare il problema con gli strumenti che conosciamo!

.. tentativi di soluzione "artigianale' (I)

========Dimensione Variabile in Una Classe==========

==========="Compilation Time"================

```
// Cosi' compila
class prova_mem {
 protected :
  int n  ;
  double myvect[10] ;
                //dimensione fissa
} ;
```

Niente di nuovo.

Dimensione prefissata.

```
// Cosi' NON compila

class prova_mem {

 protected :

  int n  ;

  double myvect[n] ; //dimensione variabile

} ;
```

L'errore e':

```
nbseve(~/LabCalcA_6)>g++ -c prova_mem.cc

In file included from prova_mem.cc:1:

prova_mem.h:7:
error: invalid use of non-static data member prova_mem::n

prova_mem.h:8: error: from this location

prova_mem.h:8:
error: array bound is not an integer constant before ] token
```

.. tentativi di soluzione "artigianale' (II)

==========Esempio di "Out of Scope"==========

1) Cosi' Funziona

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

int main() {

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

  cout << endl << " Dopo il loop " <<
          endl << "  a = " << a << endl << endl ;

return 1;

}
```

```
=====L'output e':

 Dopo il loop
  a = 4
```

2) Cosi' NON Compila :

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

int main() {

  int a ;

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

  cout << endl << " Dopo il loop " <<
          endl << "  a = " << a <<
    " b = " << b << endl << endl ;

  return 1;

}
```

```
========L'errore e':

nbacer(~/LabCalcA_6)>g++ -c
out_of_scope.cpp
out_of_scope.cpp: In function âint main():
out_of_scope.cpp:16:
error: b was not declared in this scope
```

.. tentativi di soluzione "artigianale' (III)

===ESEMPIO DI PUNTATORE A VARIABILE OUT OF SCOPE===

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

int main() {

  int a  ;

  int *p ;
  int *q ;

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

    a = i;
    int b = i;

    p = &a ;
    q = &b ;

  };   // … continua …
```

```
  cout << endl << " Dopo il loop : "  << endl ;
  cout << endl
       << "    Puntatori -- " << endl
       << "                 p = " << p <<
                    "      q = " << q << endl ;

  cout << endl << "    Valori Puntati " << endl
       << "                  *p = " << *p <<
                    "     *q = " << *q << endl ;

  return 1;

};
```

========L'output e':

```
===   Risultato giusto, ma casualmente!!!!!!
```

```
Dopo il loop :

     Puntatori --

               p = 0xbffff4f4      q = 0xbffff4e4


     Valori Puntati

              *p = 4     *q = 4
```

Allocaz. memoria a run time e compilation time (I)

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

using namespace std;

int main() {

//*****VETTORE DI DIMENSIONI FISSATE*************
  int myvect1[10] ;
  for(int i=0; i<10; i++) {
    myvect1[i] = i;
  };
  cout << endl << " ============  " << endl;
  for(int *p=myvect1; p!= & myvect1[10]; p++) {
    cout << *p << endl;
  } ;
  cout << endl << " =============  " << endl ;

  // … continua …
```

Allocaz. memoria a run time e compilation time (II)

```
//***VETTORE DI DIMENSIONI VARIABILI*************
  int nv = 15 ;
  int myvect2[nv] ;
  for(int i=0; i<nv; i++) {
   myvect2[i] = i;
  };

  cout << endl << " ===========  " << endl ;
  for(int *p=myvect2; p!=& myvect2[nv]; p++) {
    cout << *p << endl;
  } ;
  cout << endl << " =============  " << endl ;

  // … continua …
```

Allocaz. memoria a run time e compilation time (III)

```
//*****VETTORE DI DIMENSIONI VARIABILI FISSATE IN ESECUZIONE******

  int ne ;

  cout << "   Assegnare ne :  " ;
  cin  >> ne ;
  cout << endl ;

  int myvect3[ne] ;

  for(int i=0; i<ne; i++) {
    myvect3[i] = i;
  };

  cout << endl << " =============  " << endl;

  for(int *p=myvect3; p!=& myvect3[ne]; p++) {
    cout << *p << endl;
  } ;

  cout << endl << " =============  " << endl ;

  return 0;

};
```

Allocaz. memoria a run time e compilation time (IV)

```
============

0
1
2
3
4
5
6
7
8
9

 ============
```

```
============

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
============
```

```
Assegnare ne : 19
============

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18

 ============
```

Puntatore a Variabile locale (I)

=====ESEMPIO DI PUNTATORE A VARIABILE LOCALE=====

===File prova_fp.h

```
class prova_fp {
 protected:
   // .......
 public :
    prova_fp() { }  ;    // costruttore
    ~prova_fp() { } ;    // distruttore
    int * int_fp() {     // un metodo
      int a = 5 ;
      return &a ;
    };
} ;
```

Puntatore a Variabile locale (II)

===Main

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

using namespace std;

int main() {
  int * p;
  prova_fp myprova;
  p = myprova.int_fp();
  cout << endl << endl;
  cout << " Puntatore : " <<  p << endl ;
  cout << " Valore    : " << *p << endl ;
  return 1;

};
```

Puntatore a Variabile locale (III)

```
=======WARNING DI COMPILAZIONE:===========
g++ float_class.cpp -o float_class.exe
prova_fp.h: In method `int * prova_fp::int_fp()':
In file included from float_class.cpp:3:
prova_fp.h:13: warning: address of local variable 'a' returned

====OUTPUT ==========
eseguo float_class  ed ottengo

 Puntatore : 0xbffff4e0
 Valore    : -1073744648 (dipende dal compilatore)
```

Come risolverlo ? la soluzione

1. E' necessario qualcosa di nuovo!

2. L'operatore `new`
   - Uso
   - Sintassi
   - L'operatore `new` con i vettori

3. L'operatore `delete`
   - Uso
   - Sintassi
   - L'operatore `delete` con i vettori

4. Esempi

L'operatore `new`

- Uso
  - permette di istanziare un oggetto a Run Time
  - Gli oggetti creati in maniera dinamica con new non vanno *out of scope*
  - Gli oggetti creati con `new` **devono** essere distrutti **esplicitamente** con `delete`
  - Rende l'esecuzione del programma piu' lenta

- Sintassi

```
nomeclasse * myobj = new nomeclasse(...) ;
```

- L'operatore `new` con i vettori

```
nomeclasse * myvec = new nomeclasse[10] ;
```

L'operatore `delete`

- Uso
  - Gli oggetti creati con `new` **devono** essere distrutti **esplicitamente** con `delete`
  - Rende nuovamente disponibile la memoria che resterebbe inutilizzata
  - Va utilizzato in corrispondenza dell'operatore `new`
  - Ricordarsi di inserirlo nel distruttore per gli oggetti delle classi che utilizzano variabili allocate dinamicamente
- Sintassi

```
delete  myobj ;
```

- L'operatore `new` con i vettori

```
delete [] myvec ;
```

Puntatore a Variabile locale (I)

=====CON L'USO DI `new`=====

===File prova_fp_n.h

```
class prova_fp {
 protected:
   // .......
 public :
    prova_fp() { }  ;
    ~prova_fp() { } ;
    int * int_fp() {
      int * q = new int(5) ;
      return q ;
    };
} ;
```

Puntatore a Variabile locale (II)

===Main (praticamente **identico** al precedente)

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

using namespace std;

int main() {
  int * p;
  prova_fp myprova;
  p = myprova.int_fp();
  cout << endl << endl;
  cout << " Puntatore : " <<  p << endl ;
  cout << " Valore    : " << *p << endl ;
  return 1;

};
```

Puntatore a Variabile locale (III)

```
====nessun WARNING in compilazione

===L'output:
===Eseguo float_class_n ed ottengo


 Puntatore : 0x8049bc0
 Valore    : 5
```

```
int * q = new int(5) ;
```

si potrebbe anche scrivere nella forma

```
int * q = new int() ;
* q = 5 ;
```

Utilizzo di new nella allocazione di memoria a Run Time (I)

========"Dimensione Variabile in Una Classe"==========

==========**"Run Time"**================

prova_mem_n.h

```
class prova_mem {
 protected :
   double * myvect ;
 public :
   prova_mem()  ;
   ~prova_mem() ;
} ;
```

prova_mem_n.cc

```
#include "prova_mem_n.h"
 prova_mem::prova_mem() {
   myvect = new double[10] ;
          // dimensione fissata
 } ;
prova_mem::~prova_mem() {
   delete [] myvect ;
 } ;
```

Simile al caso a dimensione fissata

La memoria e' allocata a Run Time e va **cancellata esplicitamente**

prova_mem_nv.h

```
class prova_mem {
 protected :
   int n ;
   double * myvect ;
 public :
   prova_mem(int n_now) ;
   ~prova_mem() ;
} ;
```

prova_mem_n.vcc

```
#include "prova_mem_nv.h"
 prova_mem::prova_mem(int n_now) {
   n = n_now ;
   myvect = new double[n] ;
 } ;
prova_mem::~prova_mem() {
   delete [] myvect ;
 } ;
```

La dimensione di myvect e' definita dinamicamente

La memoria e' allocata a Run Time e va **cancellata esplicitamente**

Allocazione dinamica della memoria (C/C++) - (testo § 10.2.2)

- La memoria puo' essere riservata
  - staticamente: in fase di compilazione
  - dinamicamente: in fase di esecuzione del programma
- Vantaggi dell'allocazione dinamica
  - scelta della dimensione degli array in fase di esecuzione
  - lo spazio riservato non si libera automaticamente (no *out of scope*)
- Svantaggi
  - lentezza
  - responsabilita' del programmatore
    - nel verificare che la memoria richiesta sia effettivamente disponibile (solo in C, dove **la sintassi e' diversa**)
    - nel liberare la memoria non piu' necessaria

12) Classe SistemaSolare (seconda parte)

La Classe SistemaSolare: Attributi e Metodi

SistemaSolare

CorpoCeleste ** pianeti
int N

SistemaSolare(int n)

~SistemaSolare()

int aggiungiPianeta(CorpoCeleste * unPianeta)

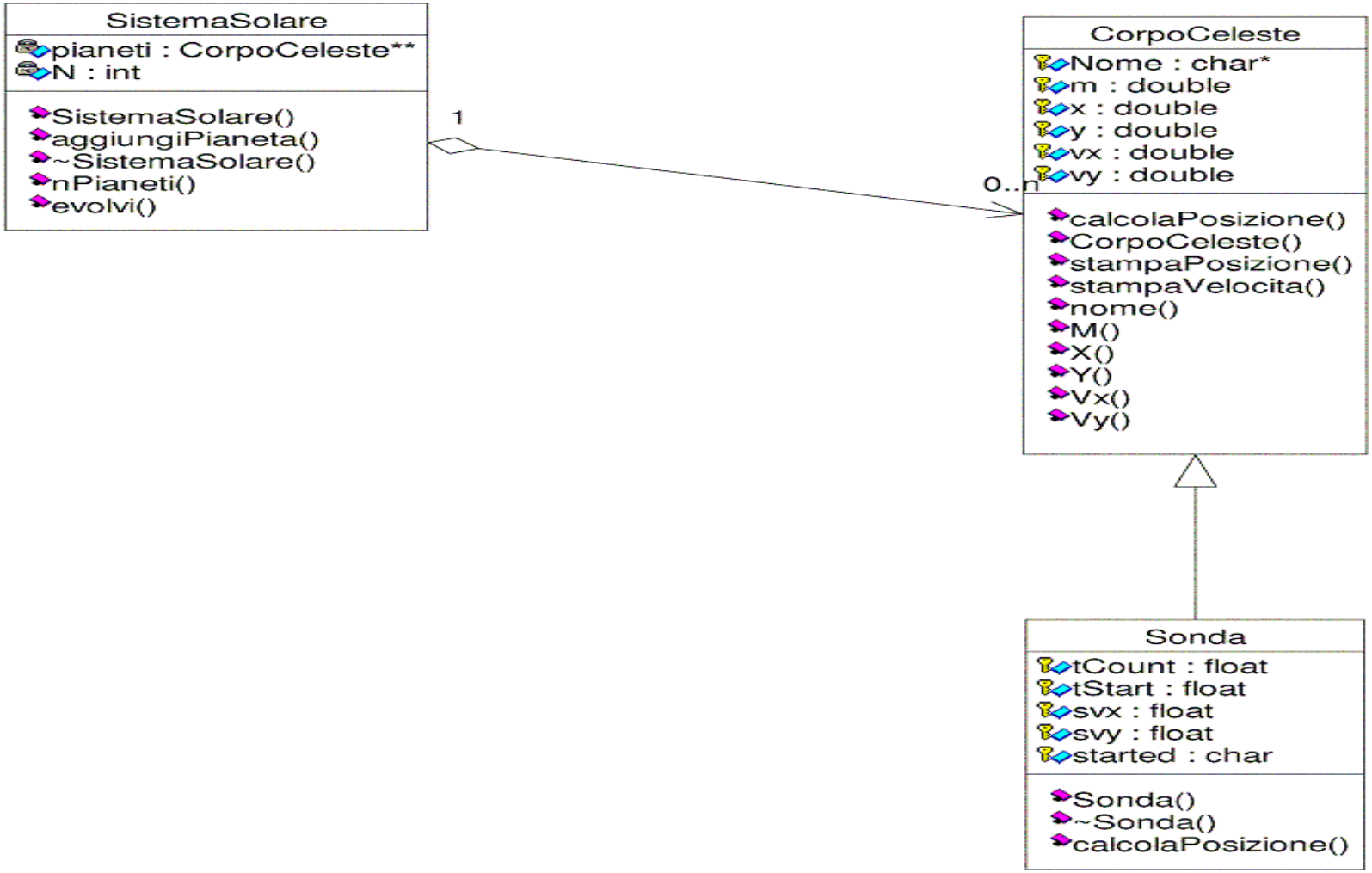
void evolvi(float T, float dt)

int nPianeti()

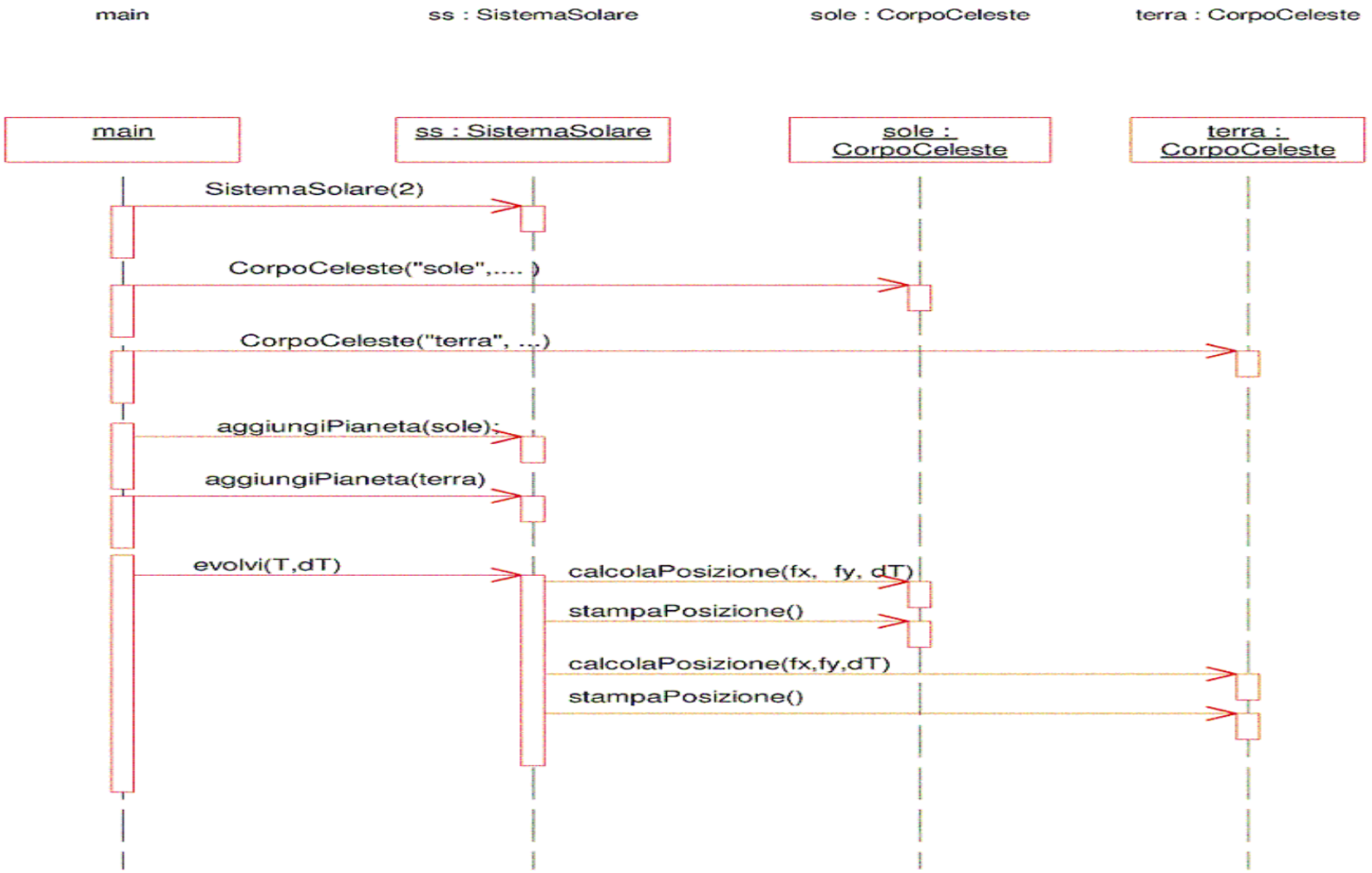
SistemaSolare.h

SistemaSolare.cc

Ricordando la Struttura ……



e l'*Interaction-diagram* della simulazione



SistemaSolare.h

```
#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 (I)

```
#include "SistemaSolare.h"
#include <cstdlib>
#include <cmath>
#include <iostream>
using namespace std ;

SistemaSolare::SistemaSolare(int n) {
   pianeti = new CorpoCeleste*[n];
       // si alloca un vettore di puntatori
       // a oggetti di tipo CorpoCeleste
       // n puntatori a CorpoCeleste
    N  = 0;
       // si pone il numero iniziale di
      //  pianeti pari a 0
}

SistemaSolare::~SistemaSolare() {
    delete [] pianeti;
}
```

SistemaSolare.cc (II)

```
int SistemaSolare::aggiungiPianeta
            (CorpoCeleste *unPianeta) {
   pianeti[N++] = unPianeta;
       // si aggiunge unPianeta alla lista dei
      // pianeti e si incrementa N di uno.
    return N;
}

void SistemaSolare::evolvi(float T, float dt) {
  float t = 0 ; //tempo dall'inizio della simulazione

    // ripeti fino a che t<=T
    while (t <= T) {
```

SistemaSolare.cc (III)

```
// loop sui pianeti
for (int i=0; i<N; i++) {

   double fx = 0.;
   double fy = 0.;

    // calcola la forza agente sul pianeta
    // i-esimo da parte di tutti gli altri
    // pianeti j-esimi,  j=0-N

   for (int j=0; j<N; j++) {

      // calcola la distanza tra i e j

      double d = sqrt(
       (pianeti[i]->X()-pianeti[j]->X())*
       (pianeti[i]->X()-pianeti[j]->X())+
       (pianeti[i]->Y()-pianeti[j]->Y())*
       (pianeti[i]->Y()-pianeti[j]->Y())
       );
```

SistemaSolare.cc (IV)

```
        // controllo che la distanza tra
        // i pianeti non sia nulla
        if (d!=0) {
          // Somma a fx e fy agenti sull i-esimo
          // corpo la forza dovuta al corpo j-esimo
          fx +=
            -G*pianeti[i]->M()*pianeti[j]->M()*
             ( pianeti[i]->X() -
               pianeti[j]->X() ) / (d*d*d) ;
          fy +=
            -G*pianeti[i]->M()*pianeti[j]->M()*
             ( pianeti[i]->Y() -
               pianeti[j]->Y()) / (d*d*d) ;
          } // termina l'if su d!=0
      } // termina il loop sul j-esimo pianeta
```

SistemaSolare.cc (V)

```
            // ora conosco la forza che agisce sul-
            // l'i-esimo pianeta e posso invocare
            // calcolaPosizione sull'i-esimo pianeta

            pianeti[i]->calcolaPosizione
                        ((float)fx, (float)fy, dt);
            pianeti[i]->stampaPosizione();

        } // termina il loop sull'iesimo pianeta

        cout << endl;
        t += dt;  // viene incrementato il tempo

    } // conclusione del while - ripeti fino a che t<=T

} // l'implementazione del metodo evolvi e' conclusa
```

13) Il programma di simulazione (simula.cpp)

simula.cpp

```
#include "SistemaSolare.h"
#include <string>

using namespace std;

int main() {

 SistemaSolare ss(2);
 CorpoCeleste sole("Il Sole",1.98e30, 0., 0., 0., 0.) ;
 CorpoCeleste terra("La Terra",
                    5.98e24, 1.52e11, 0., 0., 29476.35) ;

 ss.aggiungiPianeta(&sole);
 ss.aggiungiPianeta(&terra);

 ss.evolvi(86400*365, 86400);

 return 0;

}
```

Qualche osservazione

- Semplicita' del programma `main()` (simula.cpp)
- Linguaggio "naturale" nell'uso del programma `main()`
- Il ruolo delle Classi
  - Le responsabilita' sono ben condivise ed equilibrate
  - I metodi sono concettualmente semplici ed immediati
  - Il metodo `evolvi(...)` di SistemaSolare e' troppo lungo

Come si puo' rendere piu' chiaro e semplice il metodo `evolvi(...)`

Una proposta

Potremmo introdurre un metodo della classe SistemaSolare che calcoli la forza tra due pianeti

```
???? SistemaSolare::forza2Pianeti(int pian_i, int pian_j) ;

???? SistemaSolare::forza2Pianeti
                (CorpoCeleste * p1, CorpoCeleste*p2) ;
```

- Sappiamo scrivere rapidamente il metodo `forza2Pianeti(…)` utilizzando il codice gia' scritto nel metodo `evolvi(…)`
- Le due forme del metodo si ottengono rapidamente una dall'altra
- Ma la forza e' un vettore? Come la gestiamo? Che "tipo" e'?

Uno schema dei metodi (traccia preliminare) (1)

```
???? SistemaSolare::forza2Pianeti(int pian_i, int pian_j) {

   return
     this->forza2Pianeti(pianeti[pian_i], pianeti[pian_j]);

}


???? SistemaSolare::forza2Pianeti
                (CorpoCeleste * p1, CorpoCeleste*p2) {

      double d = sqrt(
             (p1->X()-p2->X())*(p1->X()-p2->X())+
             (p1->Y()-p2->Y())*(p1->Y()-p2->Y()));

      double fx = 0;
      double fy = 0;
                                  // continua
```

Uno schema dei metodi (traccia preliminare) (2)

```
        fx =  -G*p1->M()*p2->M()*
                 ( p1->X() - p2->X() ) / (d*d*d) ;
        fy =  -G*p1->M()*p2->M()*
                 ( p1->Y() - p2->Y()) / (d*d*d) ;


    return ??????
}
```

- Ora abbiamo l'idea!
- L'integrazione in `SistemaSolare::evolvi(….)` e' immediata
- Ci resta da capire come gestire la forza!

Tre possibilita'

- Ricercare nelle librerie del C++ qualcosa che possa rispondere alle nostre necessita.
- Ricercare in librerie esterne qualcosa che soddisfi le nostre richieste
- Scrivere una Classe che
  - ci fornisca quanto richiesto
  - non abbia tutte le *features* che potrebbero essere logicamente attribuite alla classe (e che troveremmo nei prodotti professionali) ma che noi non usiamo