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++

Pagina Web del Corso

http://webusers.fis.uniroma3.it/~bussino/Lab_Prog_Calc.html

La Pagina Web verra' aggiornata durante lo svolgimento del corso

Esercitazioni di Laboratorio

Le Esercitazioni sono iniziate

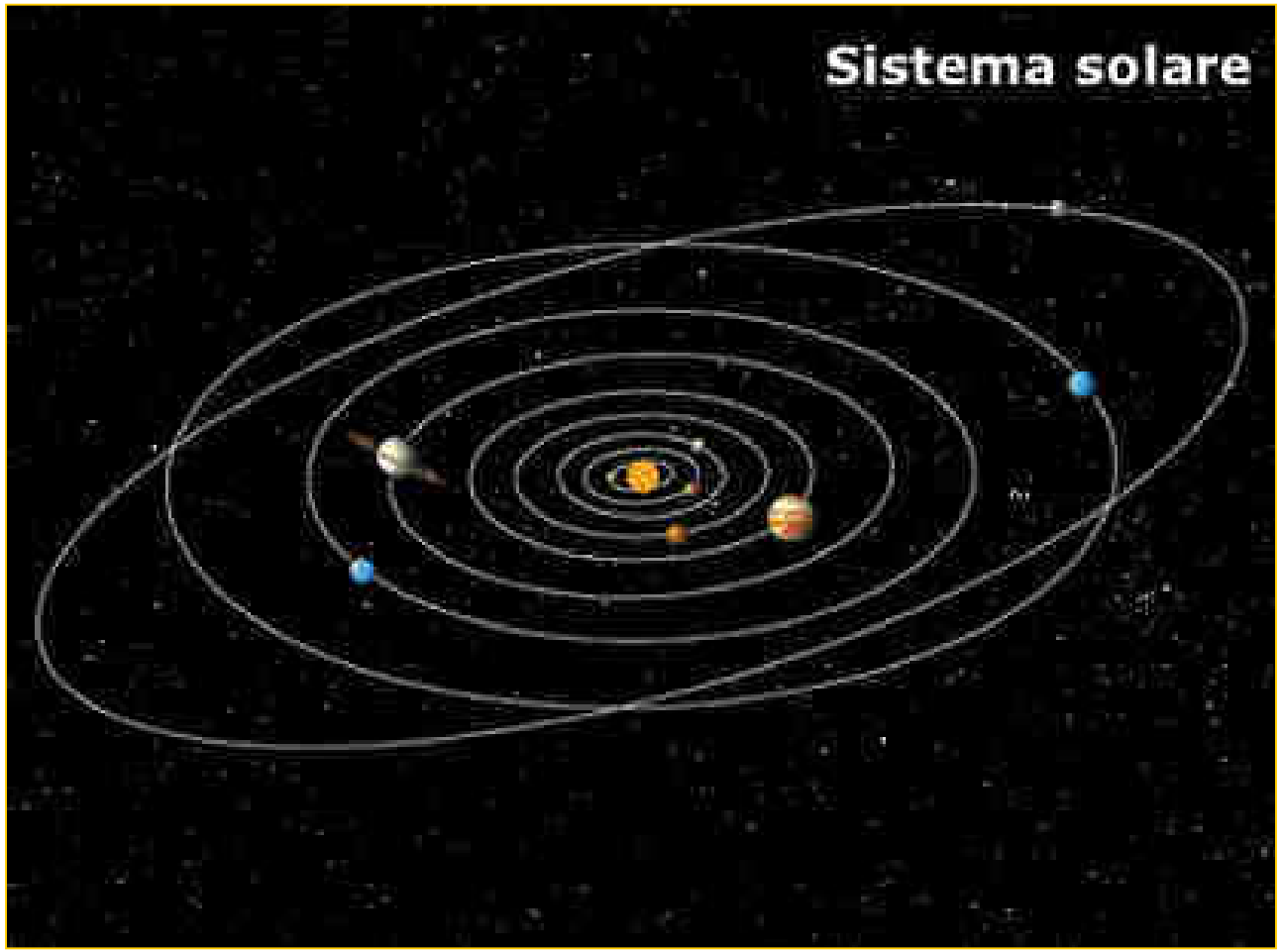
il 28 settembre per il gr. 1
il 29 settembre per il gr. 2
il 30 settembre per il gr. 3

Laboratorio di Calcolo
(Laboratori Didattici)

Via della Vasca Navale, 84

4) Simulazione del Sistema Solare

Un esempio guida (1)



Un esempio guida (2)

1. Incapsulamento
2. Ereditarieta'
3. Polimorfismo

La Simulazione di un Sistema Solare

- Il problema fisico
- Progettazione del programma di simulazione
- L'interazione tra le classi nella simulazione

Incapsulamento → Oggetti

La Simulazione di un Sistema Solare (1)

1. Il problema fisico

$$\vec{F} = m\vec{a}$$

$$\vec{F}_{12} = -G\frac{m_1 m_2}{r^3}\vec{r}$$

La Simulazione di un Sistema Solare (2)

2. Progettazione del programma di simulazione

Gli oggetti: corpi celesti, sonde, pianeti,
stelle, satelliti,…..

Le relazioni tra gli oggetti:
un pianeta e' un corpo celeste
una sonda e' un satellite…
… ed e' anche un corpo celeste

Altri oggetti mancano nella lista precedente?
Sistema Solare?
... e un Sistema Solare ha dei Corpi Celesti!

La Simulazione di un Sistema Solare (3)

3. L'interazione tra le classi nella simulazione

Quali compiti ha ciascun oggetto?

Quando il sistema evolve, chi determina le nuove coordinate di ciascun corpo celeste?

Chi ha la responsabilita' di far evolvere il sistema?

La Simulazione di un Sistema Solare (4)

Possibile scelta:

- *Oggetti*

  CorpoCeleste

  SistemaSolare

  Sonda

- ***Responsabilita' degli Oggetti***

CorpoCeleste ha la responsabilita' di calcolare la sua posizione durante l'evoluzione del sistema

SistemaSolare ha la responsabilita' di far evolvere il sistema e di determinare il nuovo stato del sistema dopo un intervallo di tempo T

SistemaSolare si occupera' di chiedere a ciascun pianeta di calcolare la sua nuova posizione dopo un intervallo di tempo ΔT.

Le Classi nella Programmazione ad Oggetti

1. Incapsulamento
2. Ereditarieta'
3. Polimorfismo

Incapsulamento → Oggetti

Oggetti → Classi

Parte successiva di questa lezione

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

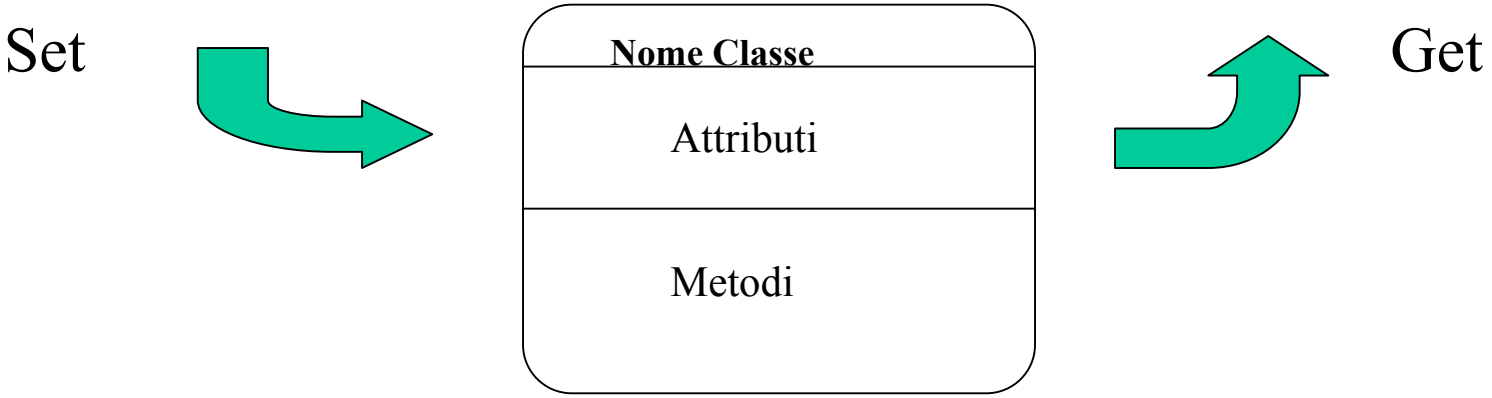
Caratteristiche di una Classe (1)

1. Attributi
   - `public`
   - **`private`**
   - `protected`

2. Metodi
   - **`public`**
   - `private`
   - `protected`

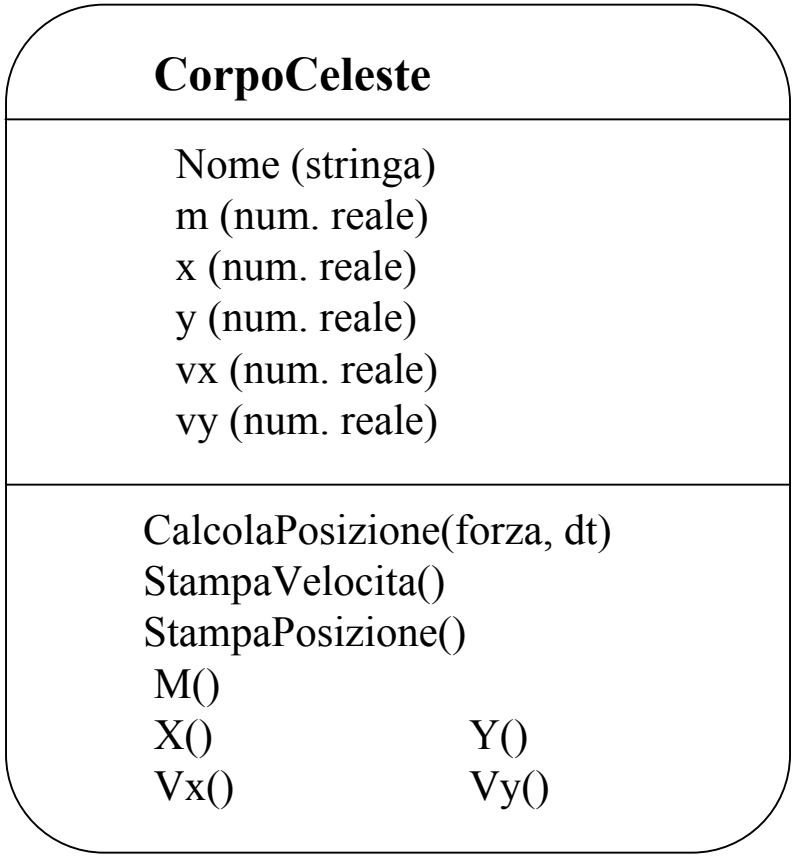
Caratteristiche di una Classe (2)

3. Accesso agli attributi di una classe
   (tipo `Set` e tipo `Get`)

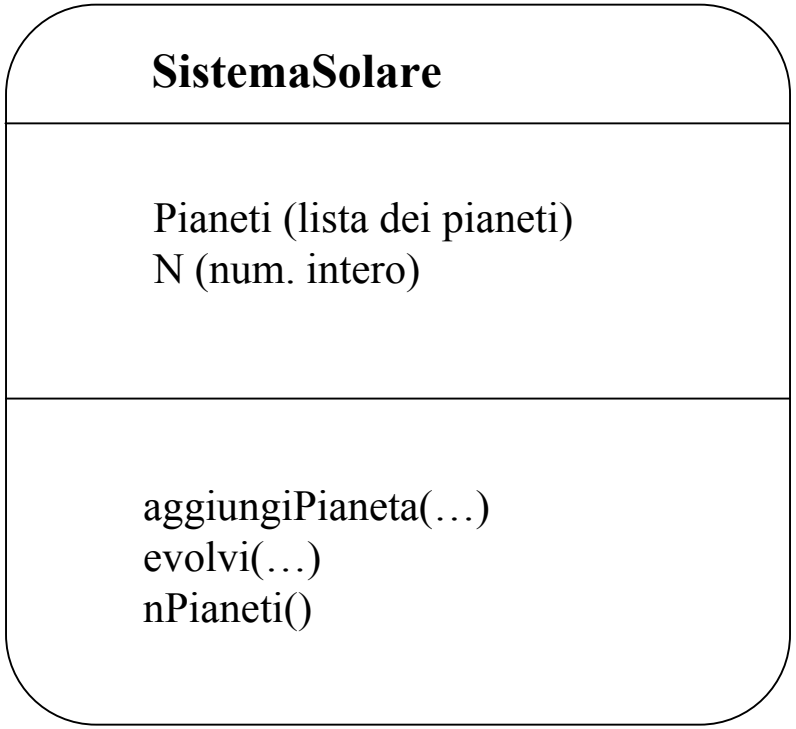


Incapsulamento

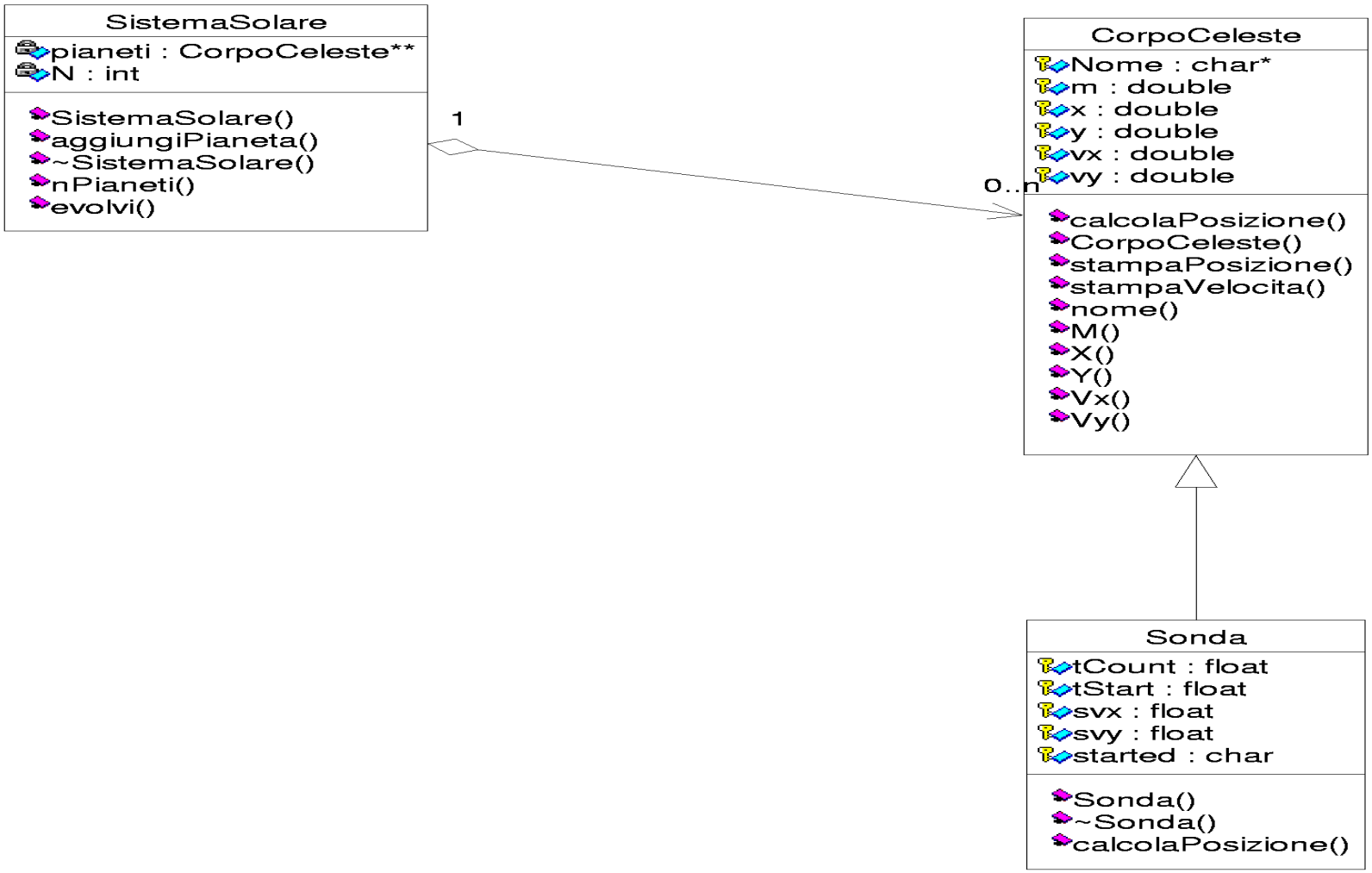
Esempio: la Classe CorpoCeleste



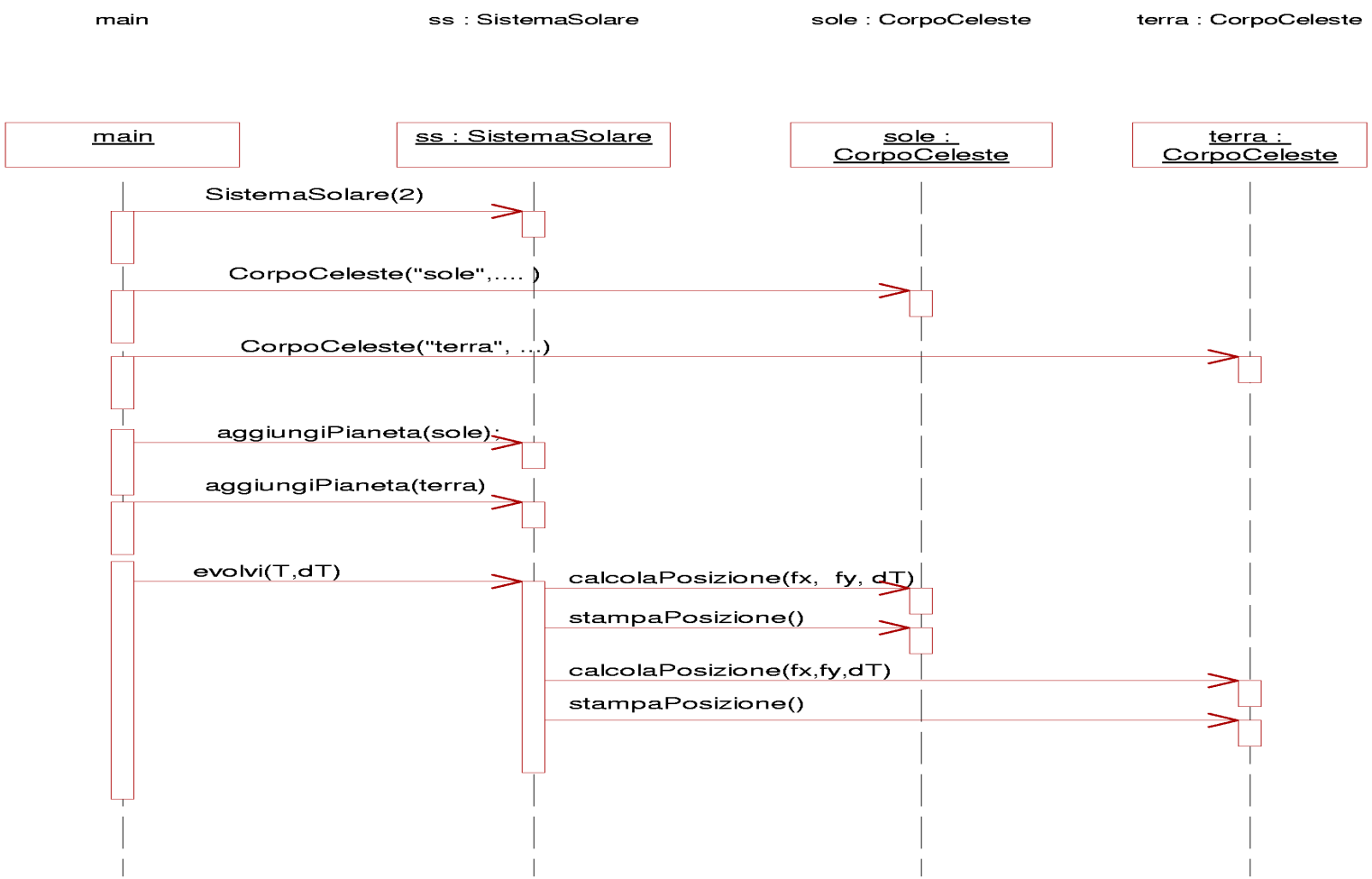
Esempio: la Classe SistemaSolare



Struttura UML del Programma di Simulazione



Interaction Diagram del Programma di Simulazione



0) Struttura del Corso

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

2) Sistemi operativi

3) Introduzione alla
Programmazione ad oggetti (OO)

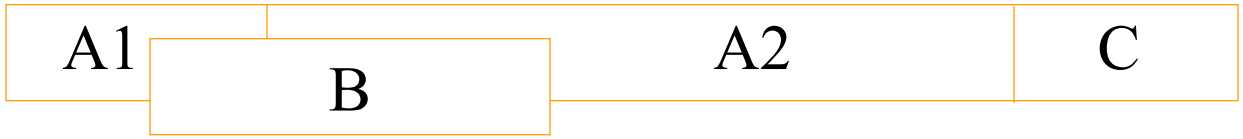
4) Simulazione del Sistema Solare

5) Introduzione al linguaggio C/C++

A – Implementazioni delle Classi in C++

B – Elementi di sintassi del linguaggio C

C – Compilazione



Le Classi nella Programmazione ad Oggetti

1. Incapsulamento
2. Ereditarieta'
3. Polimorfismo

Incapsulamento → Oggetti

Oggetti → Classi

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

Esempi: le classi CorpoCeleste e Sistema Solare

CorpoCeleste

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

CalcolaPosizione(forza, dt)
StampaVelocita()
StampaPosizione()
M()
X() Y()
Vx() Vy()

SistemaSolare

Pianeti (lista dei pianeti)
N (num. intero)

aggiungiPianeta(…)
evolvi(…)
nPianeti()

"Costruzione" di una Classe

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

1. La dichiarazione di una Classe: l'*header file* (.h)

2. L'implementazione di una Classe: l'*implementation file* (.cc)

La dichiarazione di una Classe: l'*header file* (.h)

Nomeclasse.h

Contiene la dichiarazione
degli Attributi e dei Metodi

L'implementazione di una Classe: l'*implementation file* (.cc)

Nomeclasse.cc

Contiene l'implementazione
degli Attributi e dei Metodi

CorpoCeleste.h

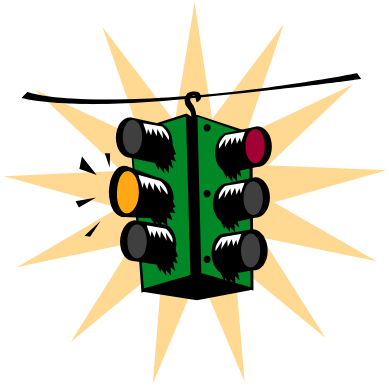
```
class CorpoCeleste {
   private:




   public:




};
```

CorpoCeleste.h

```
class CorpoCeleste {
   protected:




   public:




};
```

Come scrivere gli attributi, i metodi e la loro implementazione???

Ci manca il linguaggio!!!



Una **pausa** per imparare alcuni elementi della sintassi del linguaggio C

scegliamo un problema semplice….

… scriviamo un programma in C (stile C++)
che risolva il problema assegnato …

… dallo studio del programma ricaviamo gli elementi
fondamentali della sintassi del linguaggio C
(altri elementi nelle prossime lezioni)

una lingua moderna si impara parlando

Problema

Determinare il vostro voto finale, in trentesimi, sapendo che questo è dato da

- 4 esercitazioni che hanno un peso pari a 4/30
- un'esercitazione individuale che ha un peso di 1/3
- un test finale che ha un peso di 4/30

assumendo che tutti i voti siano espressi in trentesimi.

Il programma

- direttive del preprocessore
- simboli speciali
- parole chiave
- identificatori
- costanti

```
// questo programma calcola il voto finale
#include <iostream>
using namespace std;

int main() {

        float lab1,lab2,lab3,lab4;
        float prova,test;
        float votoFinale;

        cout << " inserire i voti"
             << " lab1 lab2 lab3 lab4 prova e test"
             << " in questo ordine"
             << endl;

        cin >> lab1 >> lab2 >> lab3 >> lab4 >> prova >> test;

        votoFinale=(lab1+lab2+lab3+lab4)*4./30.+prova/3.+test*4./30.;

        cout << " voto finale " << votoFinale << endl;

        return 1;
}
```

Simboli speciali

- Commenti
  - // all'inizio di una riga di commento
  - /* commento */
- Direttive del precompilatore
  - Tutte le righe che iniziano con il simbolo #
- Operatori
  - +, -, *, /, >>, <<, etc…
- Punteggiatura
  - Ogni istruzione C++ termina con ;
  - Ogni parte autoconsistente del programma è racchiusa tra parentesi graffe { } *scope*

Elementi del programma

- Il programma è realizzato con una sequenza di elementi base: parole chiave (*keywords*), identificatori, costanti e simboli speciali
  - Keywords: int, float, return, void etc…
  - Identificatori: main, cout, cin, lab1 etc…
  - Costanti: " voto finale " , 3. , 30. etc…
  - Simboli speciali: #, (,), +, *, <<, >> etc…

Identificatori

- Gli identificatori (nomi) degli oggetti e dei metodi sono una sequenza continua di caratteri (massimo 32) appartenenti alla lista

  'a'..'z', 'A'..'Z', '0'..'9', '_'

  con il vincolo che il primo carattere non sia un numero

- Quali di questi sono identificatori validi?

  abc, ABC, a_1, a1, m/h, 5a, voto Finale, votoFinale

- Risposta: abc, ABC, a_1, a1, votoFinale

Costanti

- Reali: 0.1 3. 3.0 1.234 -12.5 0.0 0. 1e10 5e-3
- Interi: -1 0 1 -44456 +877
- Caratteri: 'A' ' ' '3'
- Stringhe di caratteri: "qualunque cosa tra doppi apici"

Classi numeriche

Esempi di classi numeriche:

- int numeri interi che occupano una parola in memoria
- long numeri interi che usano due parole (interi lunghi)
- float numeri reali che occupano una parola
- double numeri reali che usano due parole (doppia precisione)

Gli oggetti numerici sono gli oggetti appartenenti a classi numeriche

Espressioni aritmetiche

- Le espressioni aritmetiche sono sequenze di nomi di oggetti numerici, operatori e parentesi (tonde)

- Esempi:
  - x+y
  - -x
  - x%2 // % è l'operatore modulo (resto // della divisione tra interi)
  - x*(y-z)/(x+y)

Precedenza degli operatori aritmetici

- Gli operatori *, / e % hanno precedenza su + e –
- Le espressioni vengono valutare da sinistra a destra
- Esempi:
  - 5-7*2+1 equivale a 5-(7*2)+1 e vale –8
  - 4.1/2-2 equivale a (4.1/2)-2 e vale 0.05
  - 9%5+1 equivale a (9%5)+1 e vale 5

Assegnazione

- Un'assegnazione altera il valore (stato) di un oggetto:

  identificatore_oggetto = espressione;

  Il valore dell'espressione (*rValue: right value*) viene assegnato all'oggetto (*lValue: left value*)

  lab1 = 20.0 ;

- Le assegnazioni posso essere anche concatenate

  lab1 = lab2 = lab3 = 20.0;

  (sconsigliato! Il programma risulta meno leggibile)

Esempi

- Incremento del valore di un oggetto. Le seguenti espressioni forniscono tutte lo stesso nuovo valore dell'oggetto **totale**.
  - **totale = totale + 1;**
  - totale += 1;
  - totale++;
  - ++totale;

- Decremento del valore di un oggetto. Le seguenti espressioni forniscono tutte lo stesso nuovo valore dell'oggetto **totale**.
  - **totale = totale - 1;**
  - totale -= 1;
  - totale--;
  - --totale;

Esempi

- Modifica del valore di un oggetto: anche in questo caso il valore finale di **costo** è lo stesso

  - `costo = costo + 0.2 * costo;`
  - `costo += 0.2 * costo;`
  - `costo = costo * 1.2;`
  - `costo *= 1.2;`

Conversioni di tipo

- Possiamo assegnare valori interi (int) a oggetti reali (float, double): viene usato il valore reale corrispondente (promozione)

- Quando invece assegniamo dei reali a degli interi perdiamo la parte frazionaria (troncamento)

```
int j = 1.234;
cout << j;       //   stampa  1 !

int k = -1.99;
cout << k;       //   stampa -1 !
```

Dichiarazione degli oggetti

- Ogni oggetto in un programma C++ deve essere introdotto con una dichiarazione prima di poter essere utilizzato
- Nel nostro esempio

```
float lab1,lab2,lab3,lab4;
float prova,test;
float votoFinale;
```

cin e cout sono dichiarati nel file iostream che viene incluso dall'instruzione del precompilatore

```
#include <iostream>
```

L'istruzione using namespace std rende visibili gli oggetti della libreria standard, senza la necessita' di indicarli esplicitamente (ad esempio `std::cin` oppure `std::string`)

Operazioni

- Gli oggetti di tipo **float** capiscono le operazioni +, * (ma anche -, % , etc...)
- L'oggetto **cin**, di tipo **istream**, identificato con la tastiera, capisce l'operazione >>
- L'oggetto **cout**, di tipo **ostream**, identificato con lo schermo, capisce l'operazione <<

Sintassi degli oggetti cout e cin

- `cout << espressione1 << ... << espressionen ;`

  dove ogni espressione può essere una stringa di caratteri racchiusa tra doppi apici o un'espressione che restituisca un valore numerico. Un'espressione speciale è *endl* (andata a capo)

- `cin >> oggetto1 >> ... >> oggetton ;`

  dove ogni oggetto è o un oggeto di tipo stringa o un oggetto di una classe numerica

Test dell'implementazione

```
inserire i voti lab1 lab2 lab3 lab4 prova e test in questo ordine
                0    0    0    0    0       0
votoFinale 0
```

```
inserire i voti lab1 lab2 lab3 lab4 prova e test in questo ordine
                30   30   30   30   30      30
votoFinale 30
```

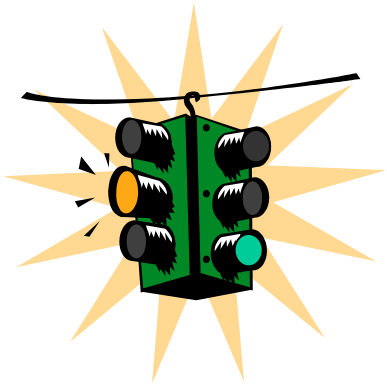
```
inserire i voti lab1 lab2 lab3 lab4 prova e test in questo ordine
                20   20   20   20   20      20
votoFinale 20
```

```
inserire i voti lab1 lab2 lab3 lab4 prova e test in questo ordine
                30   30   30   30   20      30
votoFinale 26.6667
```

NB

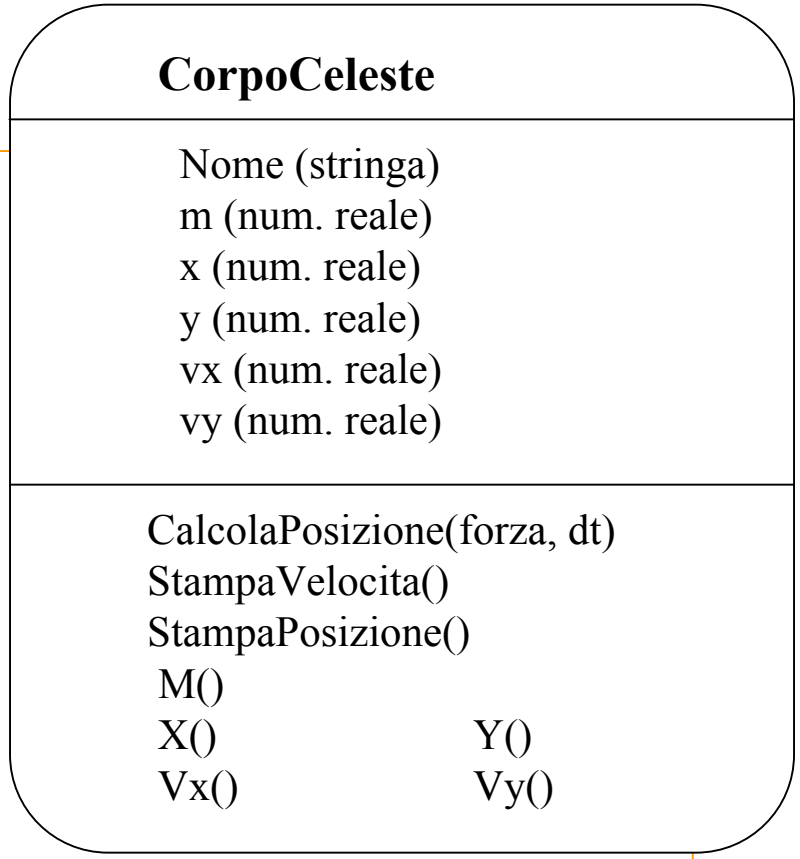
- I test rivelano solo la presenza di errori, non la loro assenza !

Ora possiamo riprendere a scrivere la dichiarazione e l'implementazione della classe CorpoCeleste



Queste nozioni preliminari dovrebbero permettervi anche di scrivere un semplice programma main che utilizzi la classe CorpoCeleste

```
class CorpoCeleste {
   protected:




   public:




};
```



CorpoCeleste.h

```
class CorpoCeleste {
  protected:
        string Nome;
        double m;
        double x;
        double y;
        double vx;
        double vy;
  public:




};
```

CorpoCeleste.h

```
class CorpoCeleste {
  protected:
       string Nome;
       double m;
       double x;
       double y;
       double vx;
       double vy;
  public:
      CorpoCeleste() ;
      CorpoCeleste(string nomeCorpo, float mass,
          float xpos, float ypos, float vxi, float vyi);
      ~CorpoCeleste() ;



};
```

CorpoCeleste.h

```
class CorpoCeleste {
  protected:
        string Nome;
        double m;
        double x;
        double y;
        double vx;
        double vy;
  public:
      CorpoCeleste() ;
      CorpoCeleste(string nomeCorpo, float mass,
          float xpos, float ypos, float vxi,float vyi);
      ~CorpoCeleste() ;
      void calcolaPosizione(float fx, float fy, float t);
      void stampaPosizione();
      void stampaVelocita();
      string nome();
      double M();
      double X();
      double Y();
      double Vx();
      double Vy();
};
```

CorpoCeleste.h

```
#ifndef  CORPOCELESTE_H
#define  CORPOCELESTE_H

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

using namespace std;


class CorpoCeleste {



                    .... (tutto come prima) ....




};

#endif
```

CorpoCeleste.cc (scheletro)

```
#include "CorpoCeleste.h"

CorpoCeleste::CorpoCeleste( ) {

}

CorpoCeleste::CorpoCeleste(string nomeCorpo, float mass,
    float xpos, float ypos, float vxi, float vyi)   {

}

CorpoCeleste::~CorpoCeleste()   {    }

void CorpoCeleste::calcolaPosizione(float fx, float fy, float t) {

}

void CorpoCeleste::stampaPosizione() {

}

void CorpoCeleste::stampaVelocita() {

}

string CorpoCeleste::nome()        {          }
double CorpoCeleste::M()           {          }
double CorpoCeleste::X()           {          }
double CorpoCeleste::Y()           {          }
double CorpoCeleste::Vx()          {          }
double CorpoCeleste::Vy()          {          }
```

CorpoCeleste.cc (1)

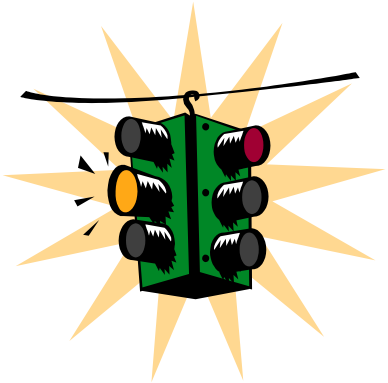
```
#include "CorpoCeleste.h"
#include <string>
#include <iostream>
#include <iomanip>

using namespace std;

CorpoCeleste::CorpoCeleste() {    }

CorpoCeleste::CorpoCeleste(string nomeCorpo, float mass,
    float xpos, float ypos, float vxi, float vyi) {

    Nome = nomeCorpo;

    m  = mass ;
    x  = xpos ;
    y  = ypos ;
    vx = vxi  ;
    vy = vyi  ;
}

void CorpoCeleste::calcolaPosizione(float fx, float fy, float t) {

    double ax = fx/m;
    double ay = fy/m;

    vx += ax*t ;
    vy += ay*t ;
    x  += vx*t ;
    y  += vy*t ;
}
```

CorpoCeleste.cc (2)

```
void CorpoCeleste::stampaPosizione() {

    cout.setf(ios::fixed);
    cout.setf(ios::showpos);
    cout << " " << setprecision(4) << setw(9) << x*1.e-11
         << " " << setprecision(4) << setw(9)  << y*1e-11 ;

}


void CorpoCeleste::stampaVelocita() {

    cout.setf(ios::fixed);
    cout.setf(ios::showpos);
    cout << " " << vx << " " << vy ;
}

CorpoCeleste::~CorpoCeleste()   {    }

string CorpoCeleste::nome()       {return Nome; }
double CorpoCeleste::M()          { return m;  }
double CorpoCeleste::X()          { return x;  }
double CorpoCeleste::Y()          { return y;  }
double CorpoCeleste::Vx()         { return vx; }
double CorpoCeleste::Vy()         { return vy; }
```

Ora abbiamo scritto la classe CorpoCeleste.

Da soli (prossima esercitazione di laboratorio) sareste certamente capaci di scrivere un programma `main` che utilizzi gli oggetti della classe CorpoCeleste



Ma come possiamo dire al computer di eseguire le istruzioni che abbiamo scritto nei nostri file (CorpoCeleste.h CorpoCelesta.cc ProvaCorpoC.cpp) ??

Compilatori ed Interpreti (1)

1. Sistemi Operativi

   Linux, Windows, Unix, FreeBSD, MS-DOS, Sun, …

   Software che sovraintende alla gestione completa del Computer (CPU, RAM, Dischi,
   Input da tastiera e Mouse, Output su video, ecc.)

2. "Pacchetti" applicativi

   Programmi (commerciali o *open source*) che eseguono una serie di operazioni per i quali sono stati progettati (Word Processing, Data-sheet, DataBase, Player, ecc.)

Sistemi operativi

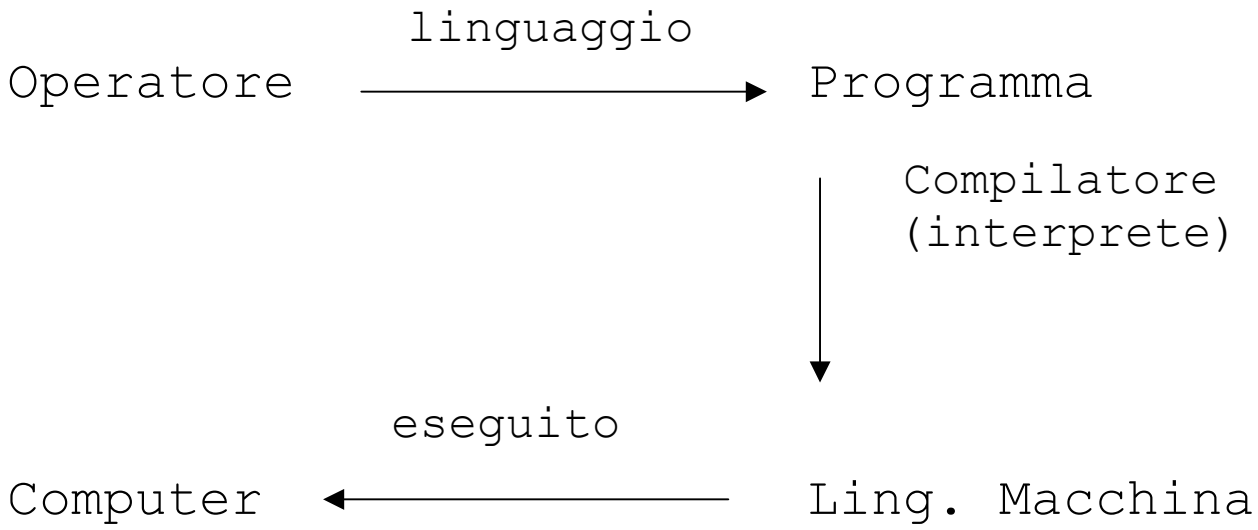
- Storicamente ogni costruttore di calcolatori ha scritto un proprio sistema operativo (*firmware*) che veniva distribuito insieme alla macchina.

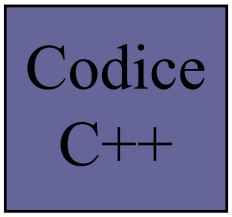
- Con l'uniformarsi delle architetture e con l'avvento dei Personal Computer, la varietà di sistemi operativi è andata riducendosi.

- Attualmente Windows è il sistema largamente più diffuso, mentre Unix è quello più utilizzato negli ambienti scientifici.

Compilatori ed Interpreti (2)

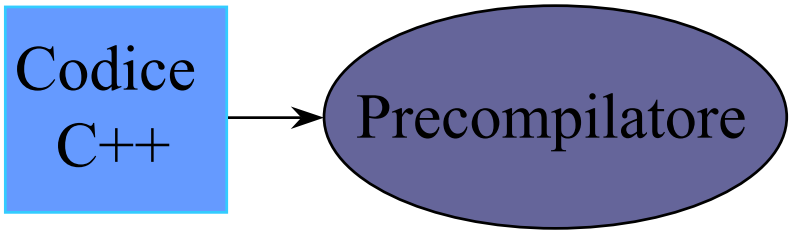
3. Compilatori (ed Interpreti)

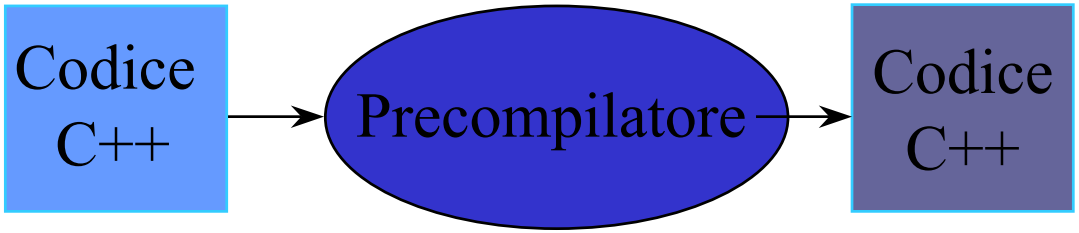



Codice
C++

Esempio di codice in C++

```
#define SCALA 2

int main(  ) {

   float x=10.;
   float y=20.;
   float z;
   z=x+y;
   z=z*SCALA;

   return 1;

}
```

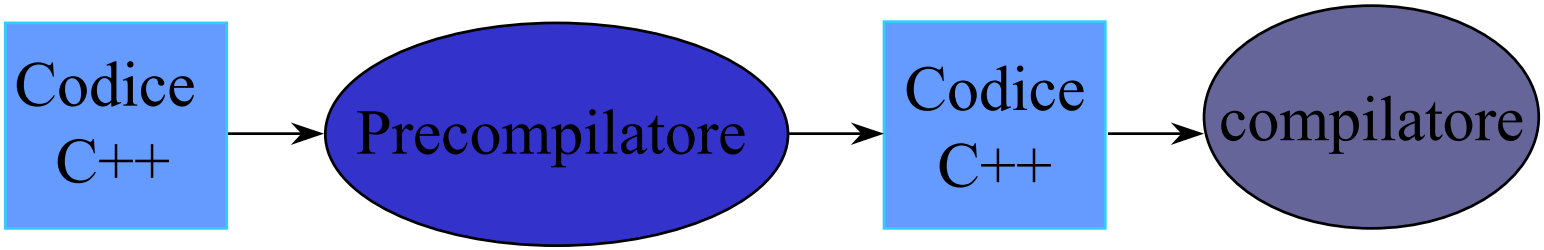

Codice
C++
Precompilatore

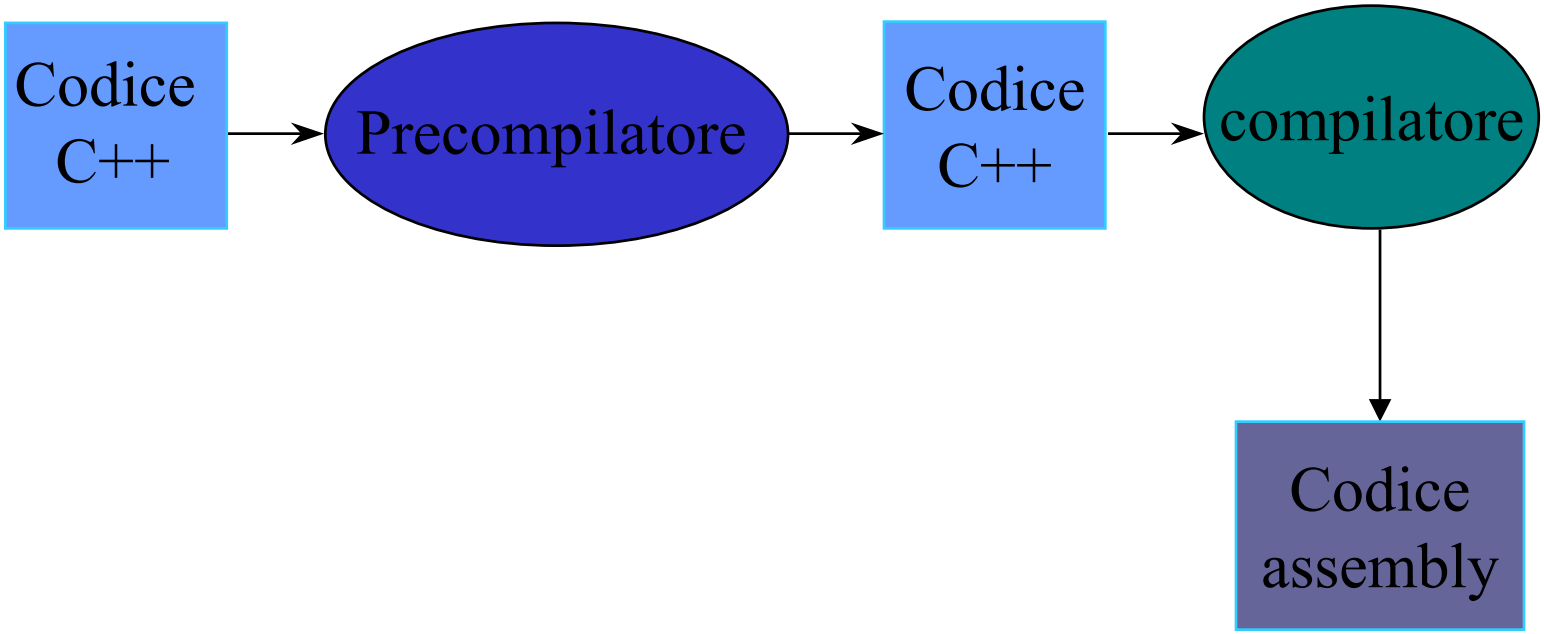

Codice
C++
Precompilatore
Codice
C++

Esempio di codice in C++ precompilato

```
#define SCALA 2

int main( ) {

    float x=10.;
    float y=20.;
    float z;
    z=x+y;
    z=z*SCALA;

    return 1;

 }
```

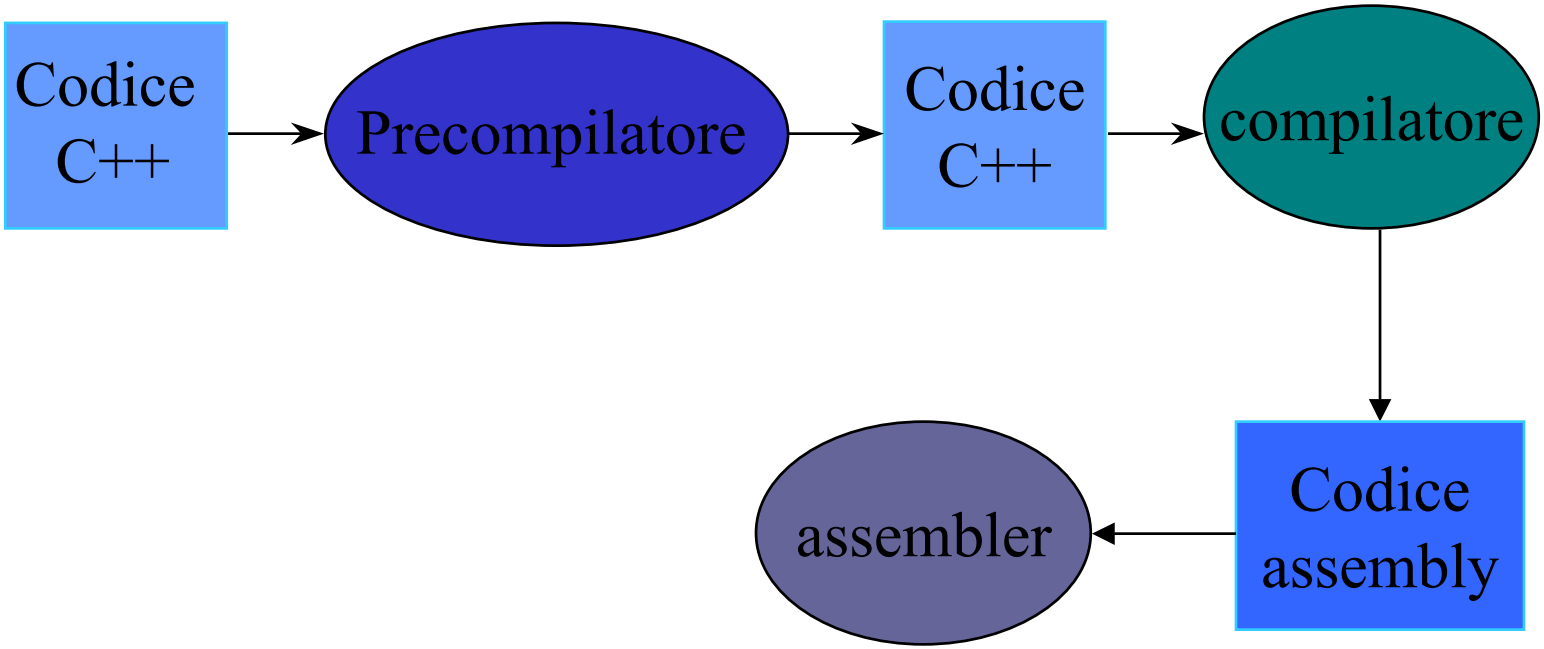
```
# 1 "preprova.cpp"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "preprova.cpp"

int main( ) {

    float x=10.;
    float y=20.;
    float z;
    z=x+y;
    z=z*2;

    return 1;

 }
```

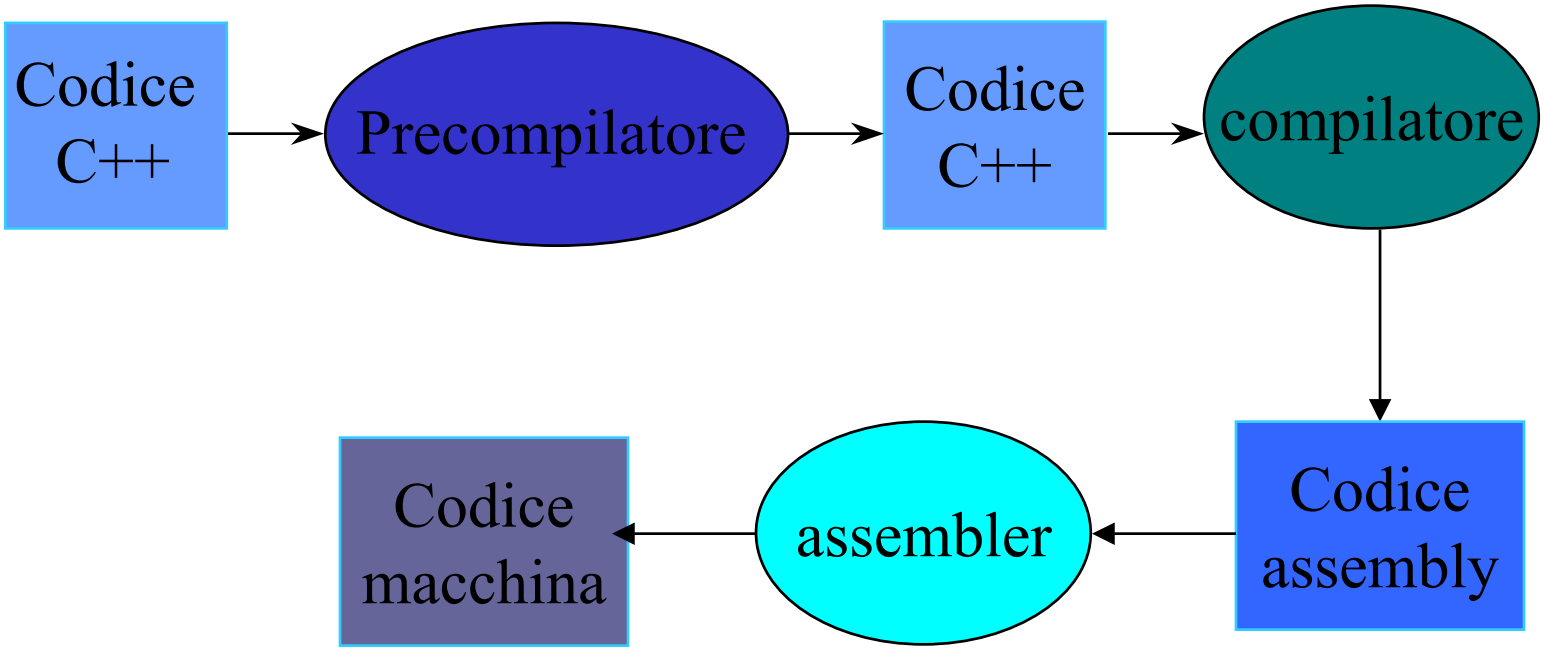

Codice
C++
Precompilatore
Codice
C++
compilatore


Codice
C++
Precompilatore
Codice
C++
compilatore
Codice
assembly

Esempio di codice assembly

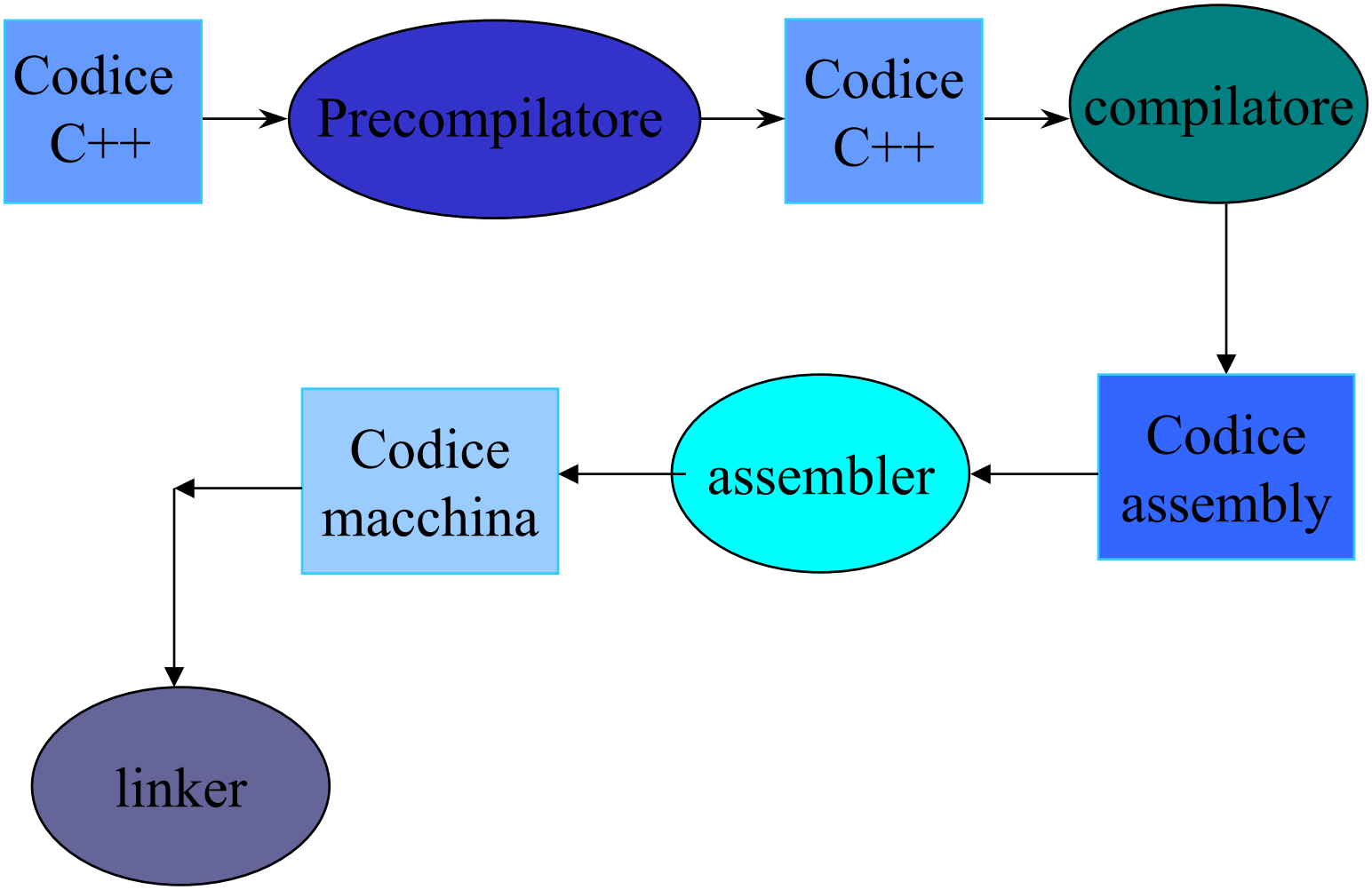
```
 .file   "preprova.cpp"
        .text
.globl main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        .cfi_personality
0x3,__gxx_personality_v0
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movl    $0x41200000, %eax
        movl    %eax, -12(%rbp)
        movl    $0x41a00000, %eax
        movl    %eax, -8(%rbp)
        movss   -12(%rbp), %xmm0
        addss   -8(%rbp), %xmm0
        movss   %xmm0, -4(%rbp)
        movss   -4(%rbp), %xmm0
        addss   %xmm0, %xmm0
        movss   %xmm0, -4(%rbp)
        movl    $1, %eax
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (GNU) 4.4.5
20110214 (Red Hat 4.4.5-6)"
        .section        .note.GNU-
stack,"",@progbits
```

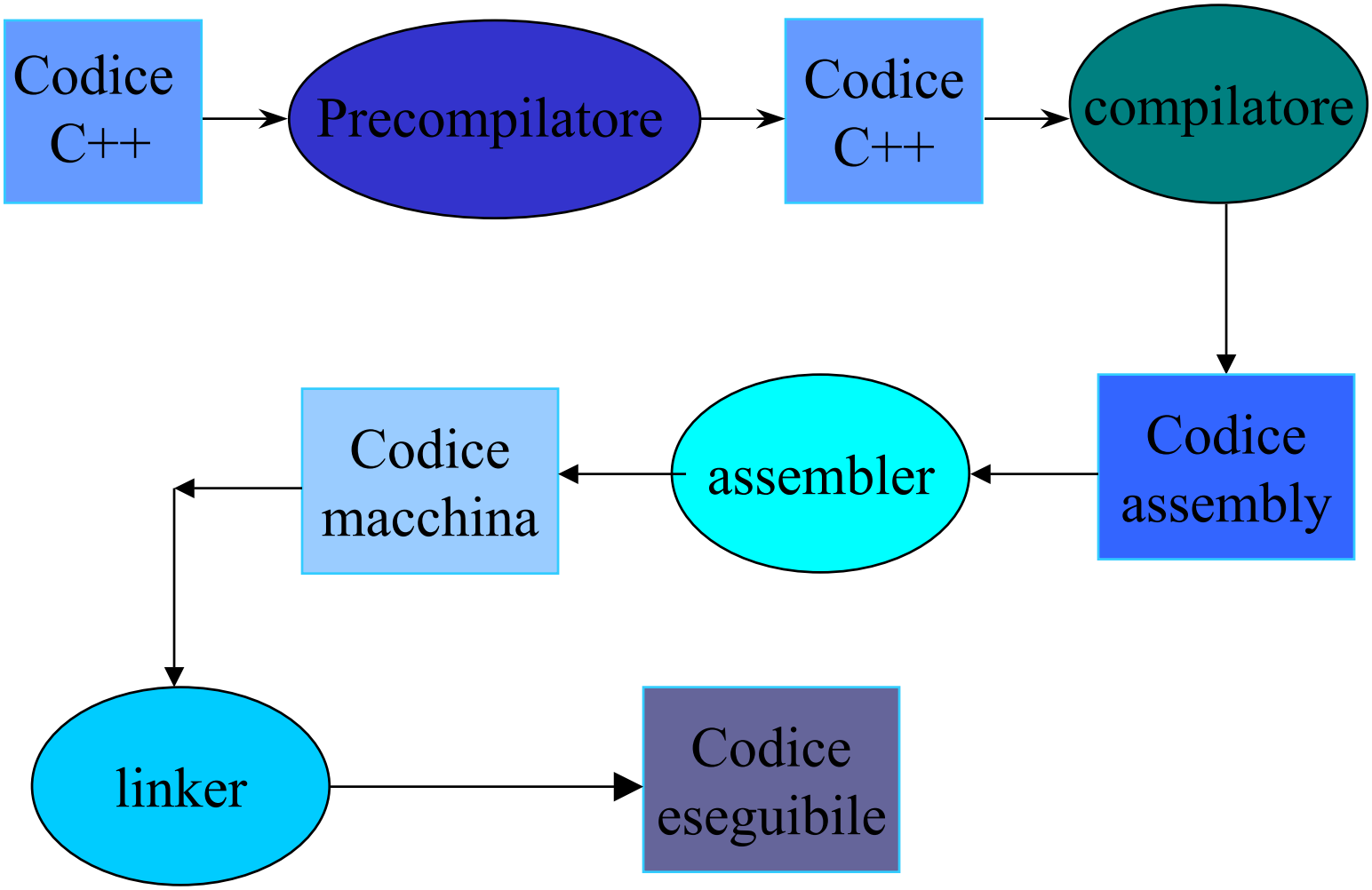

Codice
C++
Precompilatore
Codice
C++
compilatore
Codice
assembly
assembler


Codice C++
Precompilatore
Codice C++
compilatore
Codice assembly
assembler
Codice macchina

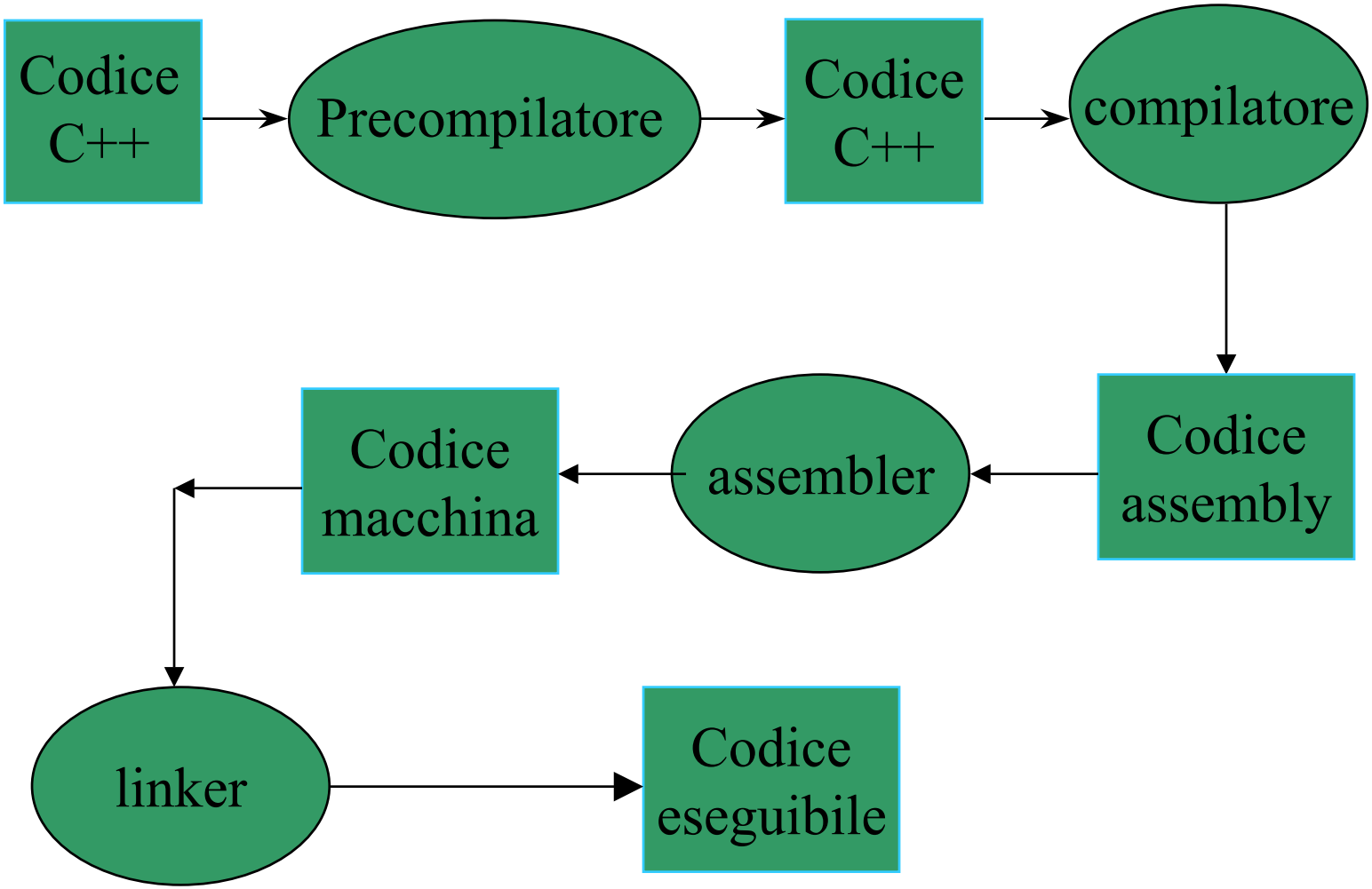
Esempio di codice macchina

10011010110111011010011010001011
10011011000001011010011010001111
11111010110001011011111011011001

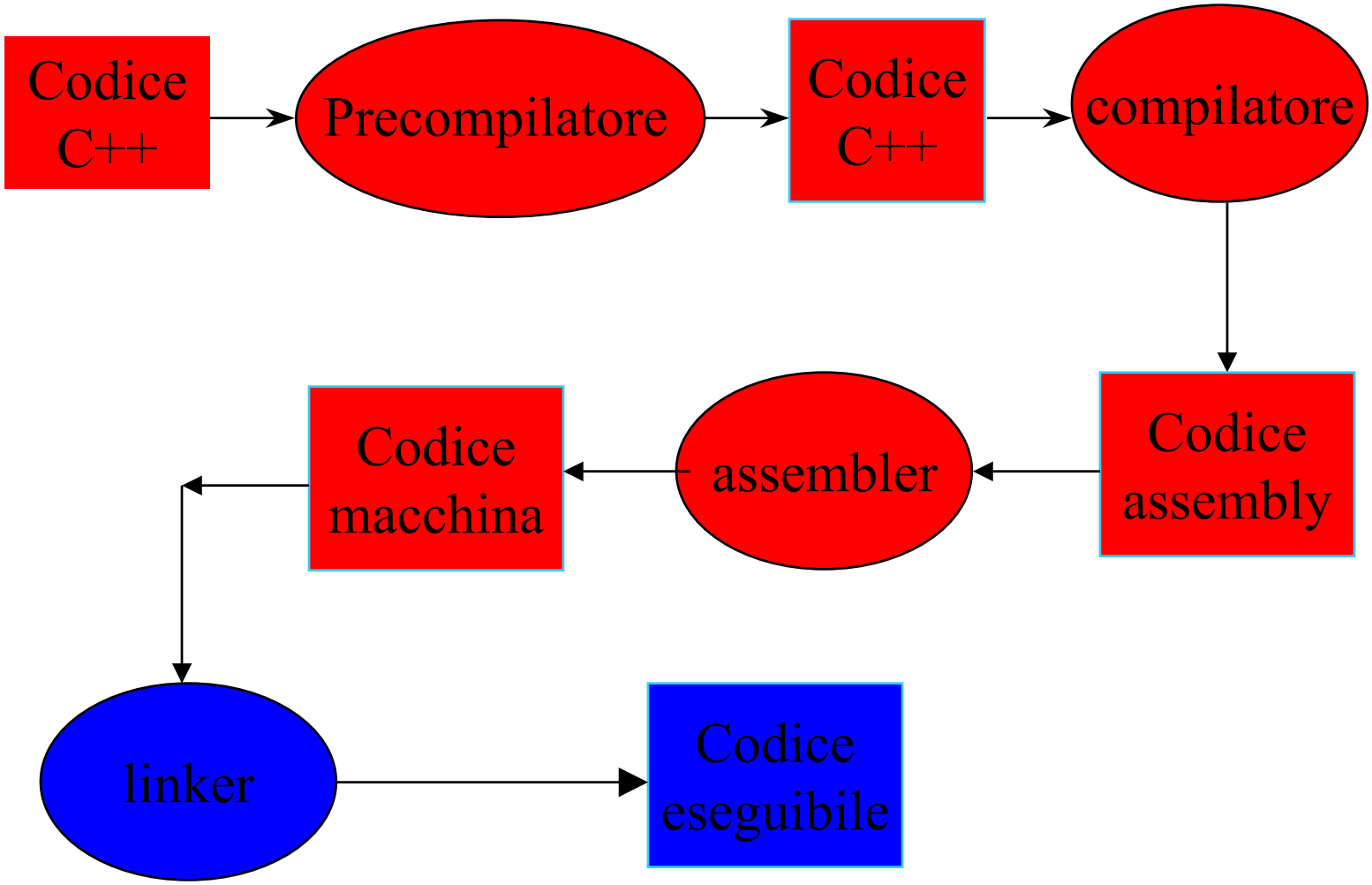

Codice C++
Precompilatore
Codice C++
compilatore
Codice assembly
assembler
Codice macchina
linker


Codice C++
Precompilatore
Codice C++
compilatore
Codice assembly
assembler
Codice macchina
linker
Codice eseguibile

Compilazione



Compilazione
Linking



Linker

- Riprendendo il nostro esempio supponiamo di volere visualizzare il risultato finale. Abbiamo già visto che a tale scopo dobbiamo usare la direttiva del precompilatore #include <iostream> per definire cout e poi dopo avere calcolato z aggiungere l'istruzione

```
cout << " risultato " << z << endl;
```

- Ma dove si trova il codice di ostream? Nella libreria (dall'inglese *library*: biblioteca) del C++ che contiene il codice macchina per tutti gli oggetti predefiniti.

- Il linker cerca automaticamente il codice cui riferito dall'utente e non presente nel codice sorgente nelle librerie di sistema.

Linker

- E se si volesse usare del codice non di sistema? Per esempio dell'altro codice già scritto dall'utente?

- Il linker accetta in input più di un file e quindi è sufficiente passagli oltre al programma utente anche gli altri file o librerie contenenti il codice che vuole riutilizzare.

Estensione dei file

- Nella maggior parte dei sistemi operativi il nome dei file è costituito da due parti separate da un punto:

  nome.estensione

- Il nome può essere un qualunque identificatore valido, l'estensione in genere identifica il tipo di file e i programmi che accettano file in input generalmente controllano l'estensione per verificarne il tipo.

- Le estensioni dipendono dal sistema operativo e dai programmi applicativi.

- In **questo corso** useremo
  - l'estensione .h per gli header file (dichiarazione di una classe)
  - l'estensione .cc per gli implementation file (implementazione di una classe)
  - l'estensione .cpp per i programmi in C++ (e in C stile C++) (altre estensioni valide potrebbero essere .cxx)

Uso di g++ (compilatore GNU)

- *g++ esempio.cpp*
  applica precompilatore, compilatore,
  assembler e linker e produce l'eseguibile a.out

- *g++ esempio.cpp -o esempio*
  chiama l'eseguibile esempio

- *g++ esempio.o altrofile.o libreria.a -o esempio*
  invoca il linker, cercando i riferimenti esterni in
  altrofile.o e in libreria.a

Uso di g++ (compilatore GNU)

Possiamo anche effettuare i singoli passaggi intermedi usando le opzioni di g++ (consultabili con *man g++*)

- *g++ -E esempio.cpp*

  invoca solo il precompilatore (e non salva l'output)

- *g++ -S esempio.cpp*

  invoca precompilatore e compilatore e salva il file assembly in esempio.s

- *g++ -c esempio.cpp*

  invoca precompilatore, compilatore e assembler e salva il file in linguaggio macchina in esempio.o

  Compilazione separata – Fortemente consigliata.

Uso di g++ (compilatore GNU)

Comando	Azione	Files accettati in input	File di output
g++ -E	Applica il precompilatore	.cc o .cpp(sorgente)	Output su schermo modificabile con l'opzione –o
g++ -s	Produce il file in assembly	.cc o .cpp (sorgente)	.s (assembly)
g++ -c	Produce il file in linguaggio macchina	.cc o .cpp (sorgente) .s (assembly)	.o (linguaggio macchina)
g++	Produce l'eseguibile	.cc o .cpp (sorgente) .s (assembly) .o (linguaggio macchina) .so o .a (librerie)	a.out , modificabile con l'opzione –o

Esempio: compilazione (separata) per la classe CorpoCeleste ed un programma di test (1)

`CorpoCeleste.h`	-	header file
`CorpoCeleste.cc`	-	implementation file
`ProvaCorpoC.cpp`	-	programma main

```
g++ -c CorpoCeleste.cc
g++ -c ProvaCorpoC.cpp
g++ ProvaCorpoC.o CorpoCeleste.o -o ProvaCorpoC
```

Esempio: compilazione (separata) per la classe CorpoCeleste ed un programma di test (2)

Talvolta (soprattutto quando si tratta di piccole modifiche al programma o correzioni successive), e' possibile unire le ultime due istruzioni in una sola

```
g++ -c CorpoCeleste.cc

g++ ProvaCorpoC.cpp CorpoCeleste.o -o ProvaCorpoC
```

Scrivere e compilare poche righe di codice alla volta!