INTRODUZIONE

Informatica - Modulo di Programmazione

Ingegneria gestionale, a.a. 2016-17

Patrizia Scandurra

Obiettivo del corso

- Lo studio della programmazione dei calcolatori elettronici
- ovvero scrivere programmi per risolvere problemi di calcolo
- Il corso non assume prerequisiti di nessun genere

Sommario

- Informazioni generali sul corso
- Linguaggi di Programmazione
  - Sintassi e Semantica
- Ciclo di Vita del Programma
  - Concetto di algoritmo
  - Compilazione e collegamento
- Ambiente di sviluppo
- Programma base

Informazioni generali

- **Sito del corso**: http://cs.unibg.it/scandurra/informatica2017.html
- **Link materiale didattico: cartella Dropbox**
  https://www.dropbox.com/sh/dkyhe897pjzgpks/AAAw6P4Zr2cCCYhEuDSq4Uc_a?dl=0
- **Ricevimento:** ??venerdì 8.45 – 11.45 ?? o per appuntamento
- **E-mail:** patrizia.scandurra@unibg.it
- **Esercitazioni in laboratorio**
  Gerardo Pelosi, Dip. di Elettronica, Informazione e Bioingegneria - DEIB, Politecnico di Milano
  - E-mails: gerardo.pelosi@polimi.it, gerardo.pelosi@unibg.it

  Inizio: verso metà Ottobre
- **Libri di testo**:
  - V. Moriggia, G. Psaila: "*Concetti fondamentali di informatica*" ed. Esculapio (Bologna: Progetto Leonardo),2007
  - P. Cremonesi, G. Psaila, "*Introduzione ragionata al C/C++*" ed. Esculapio (Bologna: Progetto Leonardo), 2000
  - G. Psaila, "*Esercizi ragionati in C/C++*" ed. Esculapio (Bologna: Progetto Leonardo), 2001

Programma

Fondamenti dell'Informatica

- Sistemi di Numerazione e rappresentazione dell'informazione.
- Architettura dei calcolatori.
- Sistemi Operativi: funzionalità ed architettura, processi, memoria, file system.
- Reti di calcolatori e internet.

Programmazione

- Concetti di base. Variabili. Input/Output. Istruzioni di Controllo.
- Vettori. Funzioni: chiamata e ricorsione.
- Puntatori, stringhe.
- Strutture dati complesse, allocazione dinamica, strutture dati dinamiche (lista semplice)

Modalità esame

- Prova scritta
  - Una prova in itinere (opzionale), vale come bonus [0,3] sulla prova scritta
- Prova orale

Linguaggi di Programmazione

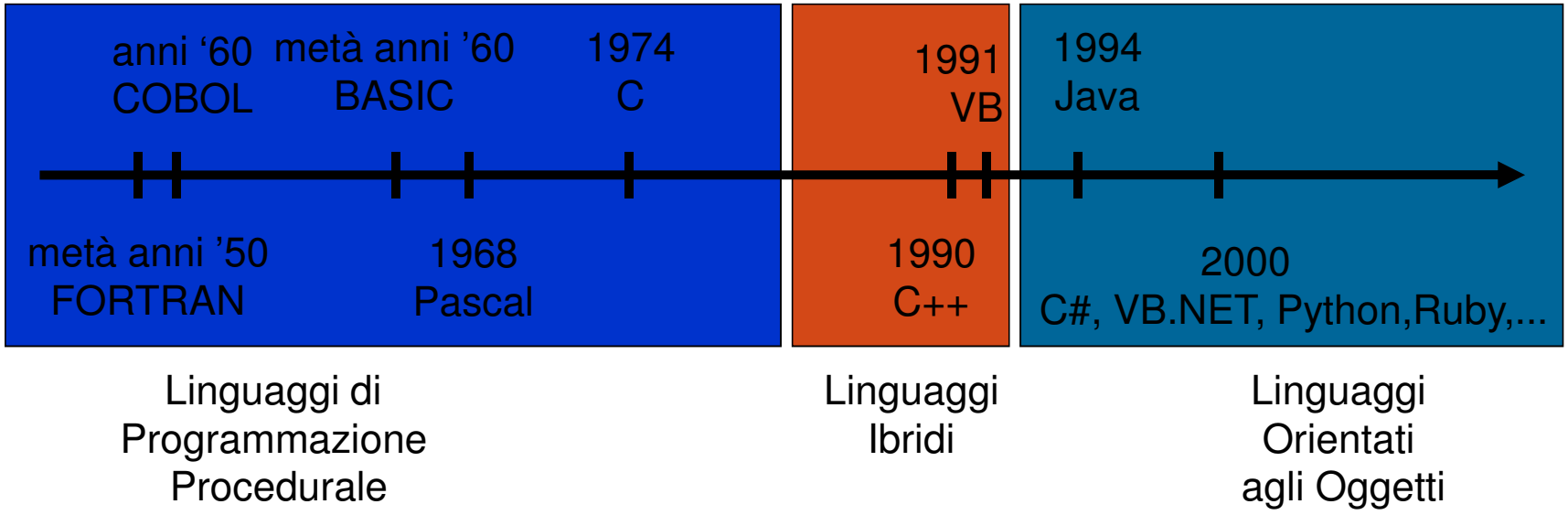
- Linguaggi per impartire istruzioni al processore
- **Programma**
  - sequenza di istruzioni
  - normalmente pensato per risolvere un problema di calcolo
  - al programma vengono forniti dei dati
  - il programma calcola eseguendo le istruzioni
  - il programma restituisce i risultati

Linguaggi di Programmazione

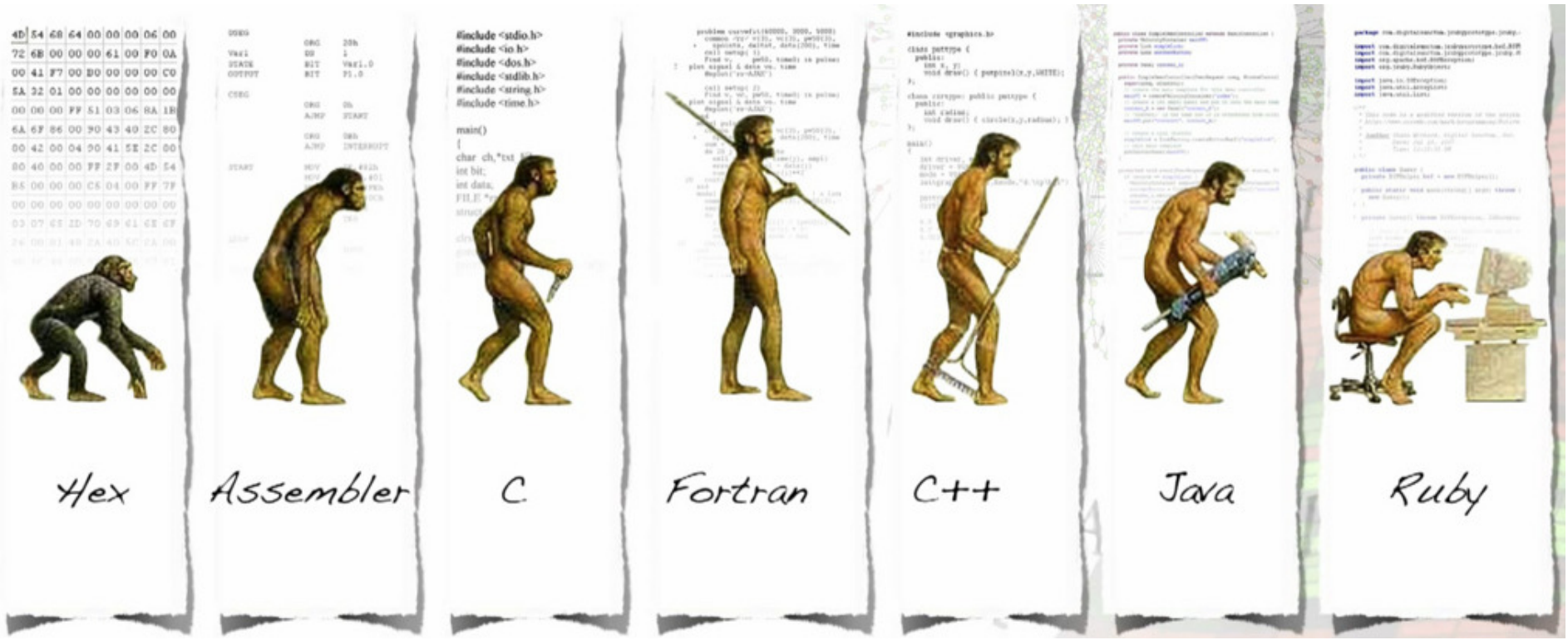
- **Linguaggi macchina**: ogni loro istruzione (vocabolo) è composta come una sequenza cifre binarie direttamente decodificabile dalla specifica macchina (CPU) per essere attuata
- **Linguaggi di alto livello**: linguisticamente più vicini al linguaggio naturale (1 istruzione equivale a 2+, o anche 10+, istruzioni in linguaggio macchina)
- **Linguaggi assembler**: linguisticamente più vicini alle istruzioni eseguite direttamente dalla macchina (CPU) (1 istruzione assembler equivale a 1 istruzione in linguaggio macchina)

Evoluzione dei linguaggi di programmazione

- Esistono numerosi linguaggi differenti per funzionalità e **paradigma**



Evoluzione dei linguaggi di programmazione



Compilatori e interpreti

- I **compilatori** sono *software* che **traducono** i programmi scritti in un linguaggio d'alto livello in **codice macchina**
  - una volta che il programma è stato *interamente* tradotto viene eseguito dal calcolatore
- il maggior vantaggio della compilazione è senz'altro l'efficienza in termini di prestazioni, al prezzo del restare **vincolati ad una piattaforma (*combinazione di architettura hardware e sistema operativo*)** particolare per poi eseguire il programma tradotto
- Esempi di linguaggi *compilati:* Cobol, C, C++, Pascal, Fortran

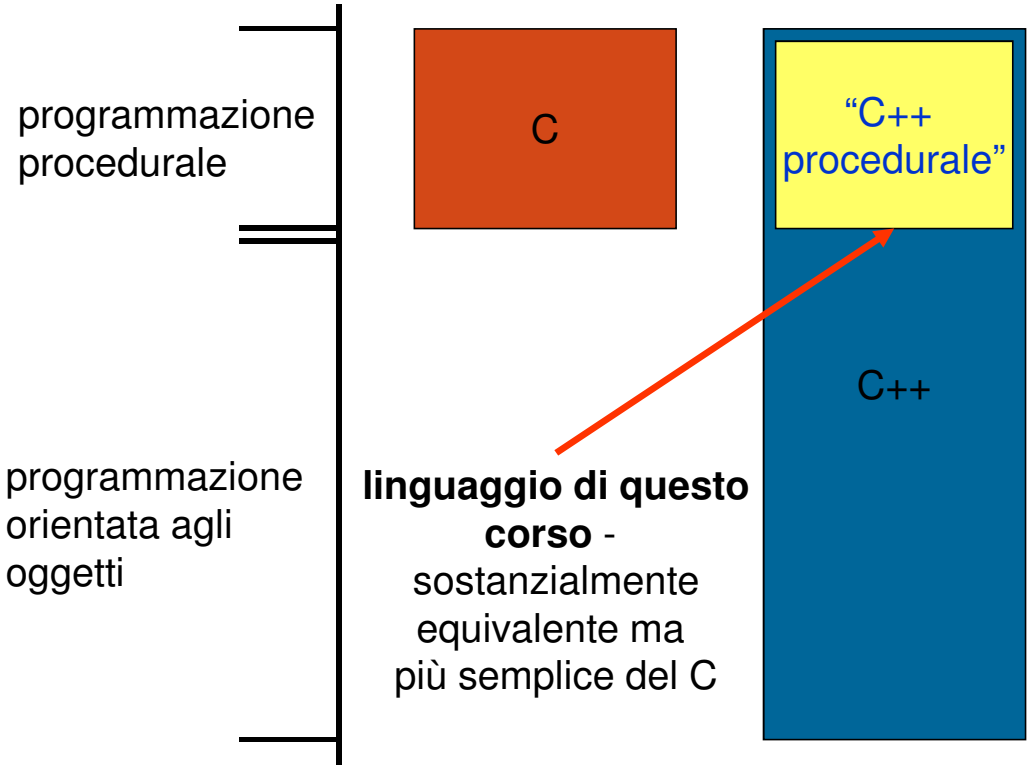
Compilatori e interpreti

- Gli **interpreti** sono programmi che **traducono ed eseguono** ciascuna istruzione del programma scritto in un linguaggio d'alto livello in modo sequenziale
- un programma scritto in un linguaggio interpretato **no**n ha, in linea di massima, **dipendenze dalla specifica piattaforma** su cui viene eseguito
- ma è più lento e richiede più memoria in fase di esecuzione
- Esempi di linguaggi interpretati: Lisp, Prolog (usati nell'intelligenza artificiale), Basic, PHP, JavaScript

Compilatori e interpreti

- **Bytecode**: una soluzione intermedia fra compilazione e interpretazione; il programma viene tradotto in un **codice intermedio** destinato a essere interpretato al momento dell'esecuzione del programma
- Vantaggi: avere la portabilità dei linguaggi interpretati grazie alla pre-compilazione, un interprete più semplice
- Introdotta nelle prime versioni del linguaggio *Pascal* e successivamente adottata nei linguaggi *Java* e *Python*, *Visual Basic (VB)* e *.NET* di Microsoft
- Esistono anche i **compilatori JIT (Just In Time)** che compilano al volo il codice intermedio e mandano in esecuzione un normale codice macchina nativo, eliminando completamente la necessità dell'interprete

Linguaggio di programmazione del corso



Il linguaggio C

- Sviluppato da Dennis Ritchie ai Bell Labs nel 1972 per realizzare il sistema operativo UNIX
- **K&R C**: 1978 (prima versione, "K&R" dal nome degli autori del libro che lo ha divulgato: "Il linguaggio C", B. Kernighan, D. Ritchie, 2 a ed., 2004, Pearson/Prentice-Hall (2 a ed. originale: 1988)
- "Testo sacro" per generazioni di programmatori, la versione italiana è aggiornata secondo l'errata corrige dagli autori per aderire allo standard ANSI 89
- **ANSI C**: 1989 (alias: Standard C, C89)
- **ISO C**: 1990 (quasi identico al C89, alias: C90)
- **C99**: 1999 (Nuovo standard ISO)
- **Standard ANSI 1989** (ancora il più utilizzato)

Il linguaggio C++

- Linguaggio di programmazione orientato agli oggetti, con tipizzazione statica
- È stato sviluppato (in origine col nome di "C con classi") da Bjarne Stroustrup ai Bell Labs nel 1983 come un miglioramento del linguaggio C
- Il C++ fu standardizzato nel 1998 (ISO/IEC 14882:1998 "Information Technology - Programming Languages - C++", aggiornato nel 2003)
- **C++11**, conosciuto anche come **C++0x**, è il nuovo standard per il linguaggio di programmazione C++ che sostituisce la revisione del 2003
- **C++14**: l'ultima versione dello standard è stata pubblicata nel 2014

Componenti di un linguaggio

- Simile al linguaggio naturale
  - sintassi e semantica
- **Sintassi**
  - "grammatica" del linguaggio
  - insieme delle regole che stabiliscono quali frasi (programmi) sono corretti
  - in Italiano: "maiuscola dopo il punto"
  - in C++: "le istruzioni si concludono con ;"

Componenti di un linguaggio

- **Semantica**
  - "significato" del linguaggio
  - insieme delle regole che stabiliscono come il calcolatore esegue i programmi corretti
  - in italiano: "andrò è un'azione nel futuro"
  - in C++: `cout << "Ciao";`

Sintassi e Semantica

- Attenzione
  - ci sono due diverse nozioni di correttezza
- **Correttezza Sintattica**
  - assenza di errori sintattici
  - il programma è eseguibile
- **Correttezza Semantica (o Logica)**
  - implica la correttezza sintattica
  - il programma risolve correttamente il problema

Sintassi e Semantica

- Verifica della correttezza sintattica
  - viene verificata dal **compilatore**
  - consente di eliminare gli errori sintattici
- Verifica della correttezza semantica
  - esecuzione e "test" del programma
  - prove di funzionamento per controllare che il programma si comporti correttamente
  - processo più complesso e delicato

Ciclo di vita di un programma

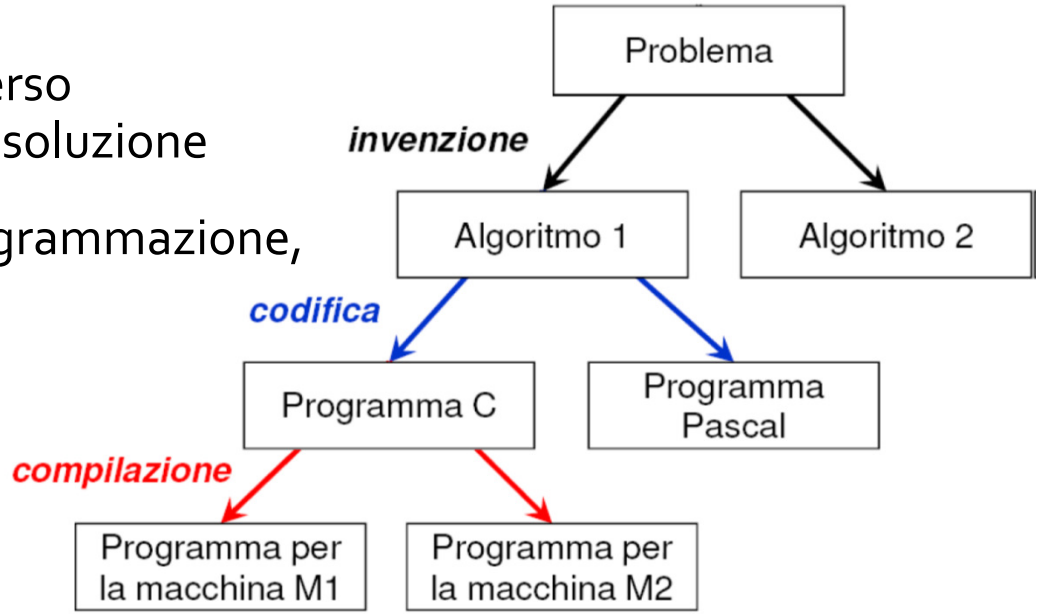
- Per programmare
  - è necessario conoscere almeno un linguaggio (sintassi e semantica)
- Ma questo non basta
  - il processo di sviluppo del software è complesso
  - è necessario un metodo (ovvero una "metodologia") per affrontare questa complessità

Ciclo di vita di un programma

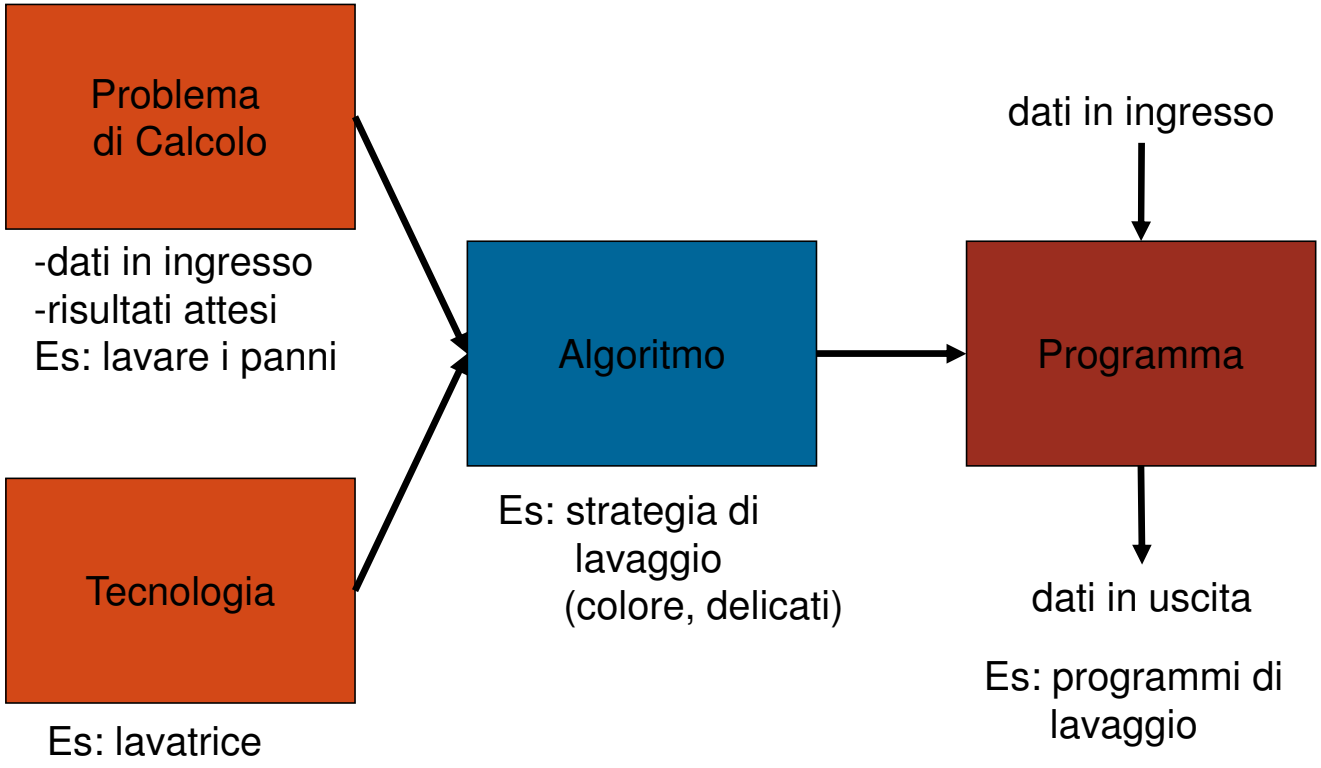
- Il punto di partenza
  - la descrizione del problema, normalmente fornita in linguaggio naturale
  - è opportuno analizzarla accuratamente
- Il punto di arrivo
  - l'applicazione correttamente funzionante
- Un passo intermedio fondamentale
  - concepire una strategia per la soluzione del problema

Il concetto di algoritmo

- **Algoritmo**
  - strategia per la soluzione del **problema**
- Il problema centrale
  - capito il problema, decidere attraverso quale sequenza di passi costruire la soluzione
- e codificarli, con un linguaggio di programmazione, in un **programma**
- Esempio: lavare i panni



Il concetto di algoritmo



Definizione di algoritmo

Un insieme finito e ben ordinato di operazioni
non ambigue ed effettivamente calcolabili che, eseguite,

producono un risultato in una quantità finita di tempo

- Esempio: algoritmo `preparaCaffè`

algoritmo `preparaCaffè`

1. Svita la caffettiera.
2. Riempi d'acqua il serbatoio della caffettiera.
3. Inserisci il filtro.
4. Riempi il filtro con la polvere di caffè.
5. Avvita la parte superiore della caffettiera.
6. Metti la caffettiera, così predisposta, su un fornello acceso.
7. Spegni il fornello quando il caffè è pronto.
8. Versa il caffè nella tazzina.

Proprietà fondamentali di un algoritmo

- **Finitezza**: la sequenza di istruzioni deve essere finita (finitezza)
- **Efficacia**: la sequenza di istruzioni deve portare ad un risultato
- **Realizzabilità**: le istruzioni devono essere eseguibili materialmente
- **Non ambiguità**: le istruzioni devono essere espresse in modo non ambiguo
- È inoltre importante valutare le **risorse utilizzate** (tempo, memoria, …) perché un consumo eccessivo delle stesse può pregiudicare la possibilità stessa di utilizzo di un algoritmo

Ciclo di vita di un programma

- Avendo concepito l'algoritmo
  - è possibile procedere alla scrittura del codice
- **Codice sorgente**
  - istruzioni del linguaggio di programmazione
  - non è direttamente eseguibile dal processore
- **Linguaggio macchina**
  - linguaggio di comandi eseguibili dal processore (molto semplici)

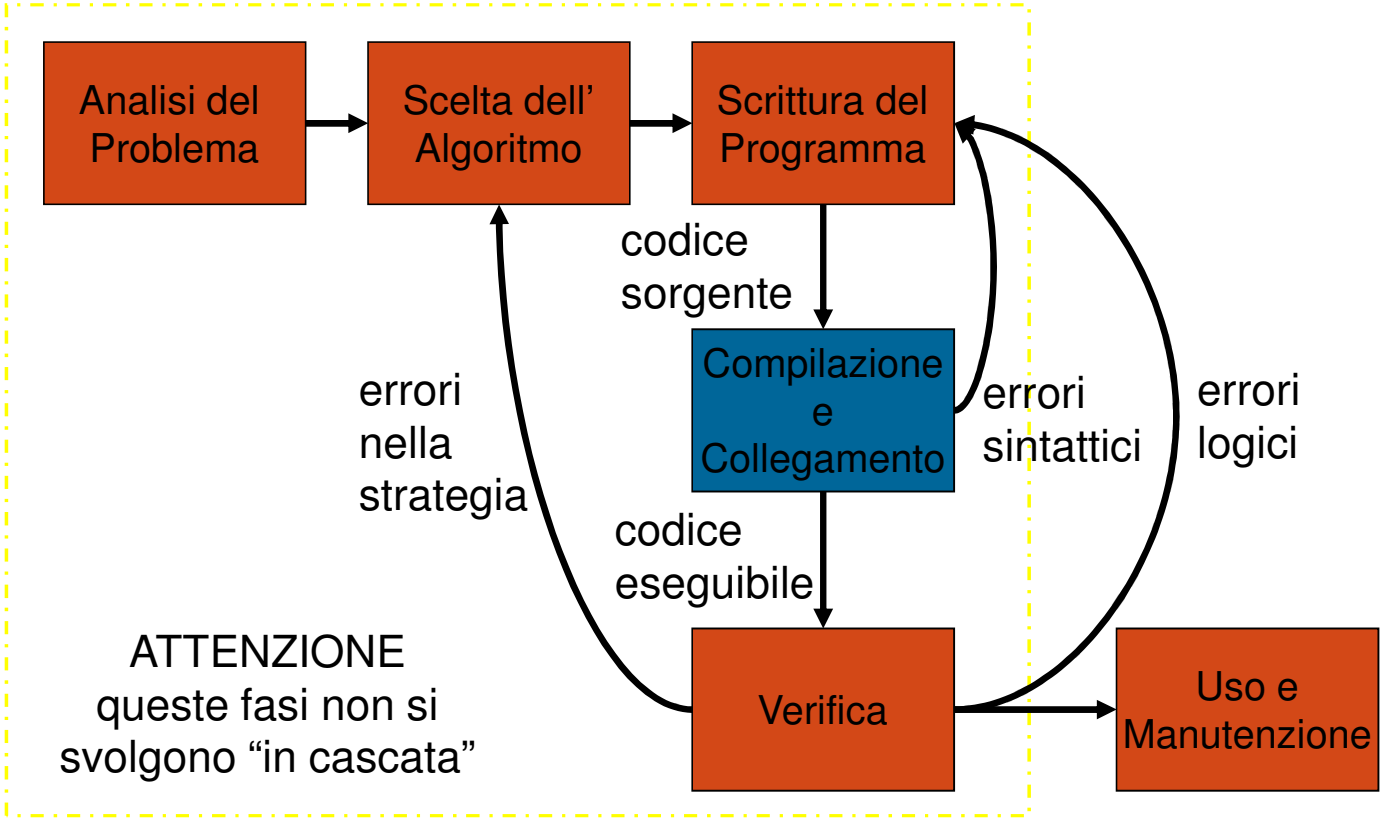
Ciclo di vita di un programma

- Processo di **compilazione**
  - verifica della correttezza sintattica
  - trasforma il codice sorgente in codice "oggetto"
- Processo di **collegamento** (**linking**)
  - collega il codice oggetto a quello delle "librerie" esterne (es: op. matematiche)
  - produce il codice eseguibile completo dell'applicazione

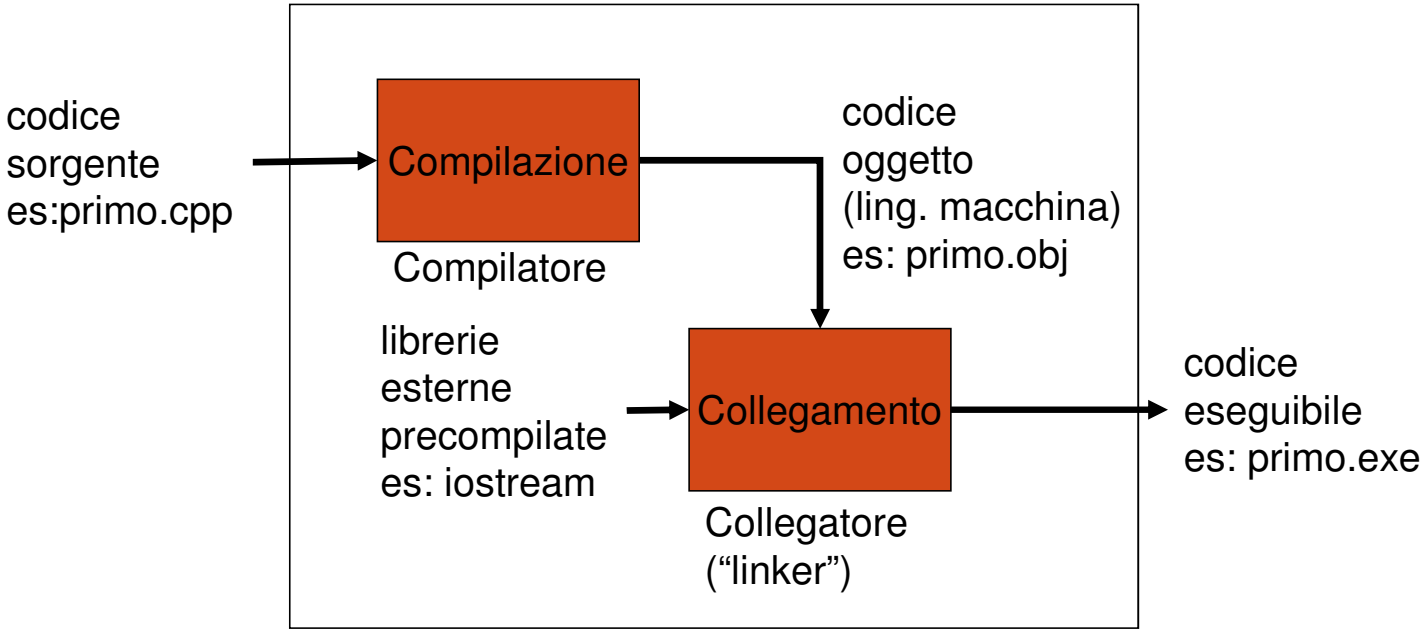
Ciclo di vita di un programma

- A questo punto
  - l'applicazione è eseguibile
  - è necessario verificarne la correttezza logica
- Fase di test
  - esecuzione ripetuta dell'applicazione su dati di test
  - per verificarne il funzionamento

Ciclo di vita di un programma



Compilazione e collegamento (linguaggi compilati)



Ambiente di sviluppo

- E' necessario disporre di vari strumenti: scrittura, compilazione ed esecuzione del programma
  - Debugger: ulteriore strumento che consente di eseguire il programma passo-passo e esaminare gli errori d'esecuzione
- **IDE = Integrated Development Environment**
- **Bloodshed Dev-C++** è l'IDE per ambiente Windows usato durante le esercitazioni /laboratorio
- Dev-C++ utilizza il compilatore **MinGW**: *porting* in Windows del compilatore **gcc** per Linux
- MinGW è un tool a riga di comando – può essere richiamato anche manualmente
  - **c:\Dev-Cpp\bin\gcc Hello.cpp -o Hello.exe**

Programma base

```
//Primo programma
//Inclusione della libreria standard di input-
 output
#include <iostream.h>


void main () {


    cout << "Hello world!" << endl;


}
```