



# ELEMENTI DI INFORMATICA

DOCENTE: FRANCESCO MARRA

INGEGNERIA CHIMICA

INGEGNERIA ELETTRICA

SCIENZE ED INGEGNERIA DEI MATERIALI

INGEGNERIA GESTIONALE DELLA LOGISTICA E DELLA PRODUZIONE

INGEGNERIA NAVALE

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II  
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE


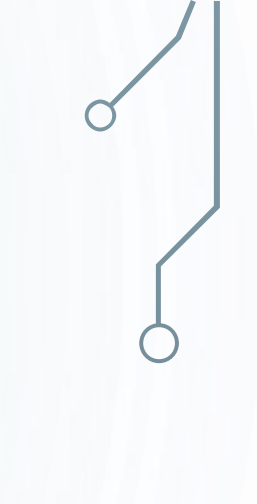
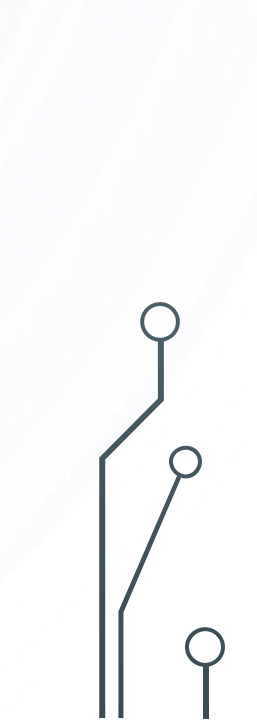
# LINGUAGGI DI PROGRAMMAZIONE

## PROGETTAZIONE DEI PROGRAMMI





# AGENDA

- Linguaggi di Programmazione
    - Basso livello
    - Alto livello
  - Paradigma di programmazione
- 
- 
- 

# LINGUAGGI DI PROGRAMMAZIONE

- Notazioni formali per descrivere algoritmi
- Dotati di un *alfabeto*, un *lessico*, una *sintassi* ed una *semantica*
- **Alfabeto**
  - Insieme di simboli costituiti da caratteri
- **Lessico**
  - Insieme di regole formali per la scrittura di insiemi di simboli dell'alfabeto, detti parole
  - Crea il vocabolario, fatto di parole leggibili e comprensibili per chi le usa
- **Sintassi**
  - Insieme di regole formali per la scrittura di frasi in un linguaggio, che stabiliscono la grammatica del linguaggio stesso
- **Semantica**
  - Insieme dei significati da attribuire alle frasi (sintatticamente corrette) costruite nel linguaggio

# LINGUAGGIO MACCHINA

- Linguaggio più semplice, direttamente compreso dalla CPU
  - Stretto legame con l'hardware
- Alta velocità di esecuzione ed ottimizzazione nell'uso delle risorse hardware
- Non trasportabilità dei programmi tra processori differenti
  - Un programma scritto per una CPU non è eseguibile da una CPU con caratteristiche diverse
- Difficoltà di programmazione
  - Gran numero di comandi per singole istruzioni
  - Istruzioni espresse sotto forma di sequenze di bit

# LINGUAGGI ASSEMBLATIVI

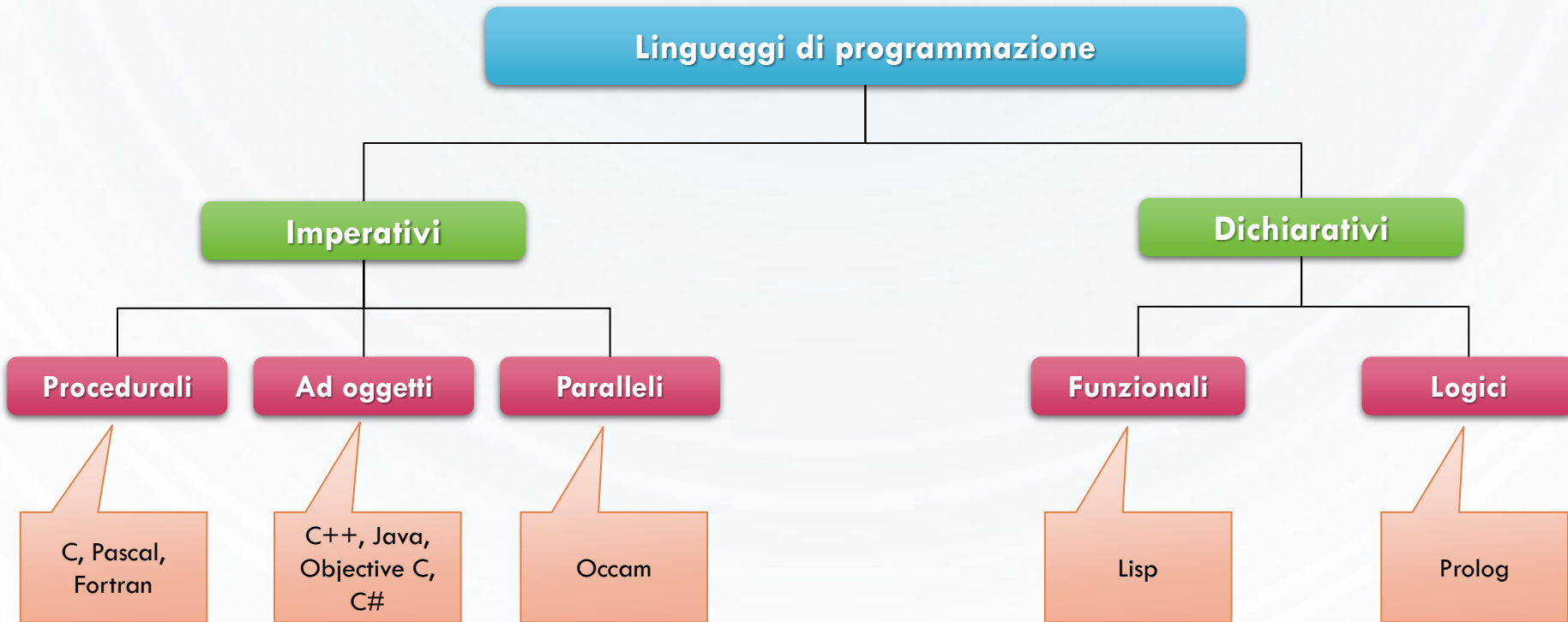
- Sostituiscono alle sequenze di bit dei codici mnemonici più facili da interpretare e ricordare
  - Mantengono uno stretto legame con le potenzialità offerte dal linguaggio macchina
- I linguaggi macchina e assembly sono detti ***linguaggi di basso livello***
  - Si pongono al livello della macchina comunicando direttamente con la CPU
  - Utilizzano codici operativi (binari o mnemonici) dello stesso processore
  - Comportano difficoltà di scrittura e di verifica del corretto funzionamento dei programmi

# LINGUAGGI AD ALTO LIVELLO

- Fanno uso di istruzioni più sintetiche e vicine al tradizionale modo di esprimere i procedimenti di calcolo da parte di un essere umano
  - Pseudo-linguaggio umano basato su parole chiave o codici operativi ispirati quasi esclusivamente alla lingua inglese
- Rendono l'attività di programmazione più semplice
- Necessitano di un "interprete" per la traduzione in reali istruzioni interpretabili dalla CPU
  - Ciascuna CPU ha il suo **traduttore** o **interprete** per garantire che lo stesso programma sia eseguito da macchine diverse
  - L'assemblatore traduce i codici mnemonici del linguaggio assembly con le sequenze di bit corrispondenti comprensibili dalla CPU

# PARADIGMA DI PROGRAMMAZIONE

- Corrisponde al modo di intendere i concetti di programma e di programmazione
- Permette di classificare i linguaggi ad alto livello





# LINGUAGGI IMPERATIVI

- Il problema viene risolto mediante una sequenza ordinata di passi
  - Sono fortemente legati al modello di Von Neumann
- Programmazione procedurale
  - Permette di descrivere le operazioni in blocchi di codice detti *sottoprogrammi*
- Programmazione ad oggetti
  - Consente la modellazione del problema come un insieme di *oggetti* che si scambiano messaggi
    - Gli oggetti sono istanze di tipi di dato astratto (le *classi*) definiti dal programmatore
  - Trasversale
    - Esistono linguaggi funzionali e logici ad oggetti, anche se è molto più frequente trovare linguaggi imperativi ad oggetti
- Programmazione parallela
  - Permette di descrivere formalmente algoritmi *non sequenziali*

# LINGUAGGI DICHIARATIVI

- Paradigma orientato al problema, cioè a **cosa** il programma deve fare per risolvere un dato problema
- Programmazione funzionale
  - Consiste nella valutazione di espressioni e nella combinazione di funzioni matematiche per generare funzioni più potenti
- Programmazione logica
  - Permette di descrivere la **struttura logica** del problema mediante definizioni ed affermazioni, piuttosto che il modo di risolverlo
  - L'esecuzione del programma consiste nell'utilizzare le informazioni ricevute per rispondere alle interrogazioni dell'utente utilizzando regole di deduzione della logica classica

# PRODUZIONE DEL SOFTWARE

- La progettazione degli algoritmi
  - È una onerosa attività intellettuale che richiede creatività ed intuito
  - È più onerosa della codifica dell'algoritmo con un linguaggio di programmazione
  - Richiede la valutazione
    - della **complessità computazionale** per il miglior utilizzo delle risorse disponibili
    - della **correttezza** per l'aderenza della soluzione alle specifiche del problema
- Il processo di produzione del software
  - non può essere di tipo artigianale, basato sulle esperienze e/o sulle iniziative del programmatore

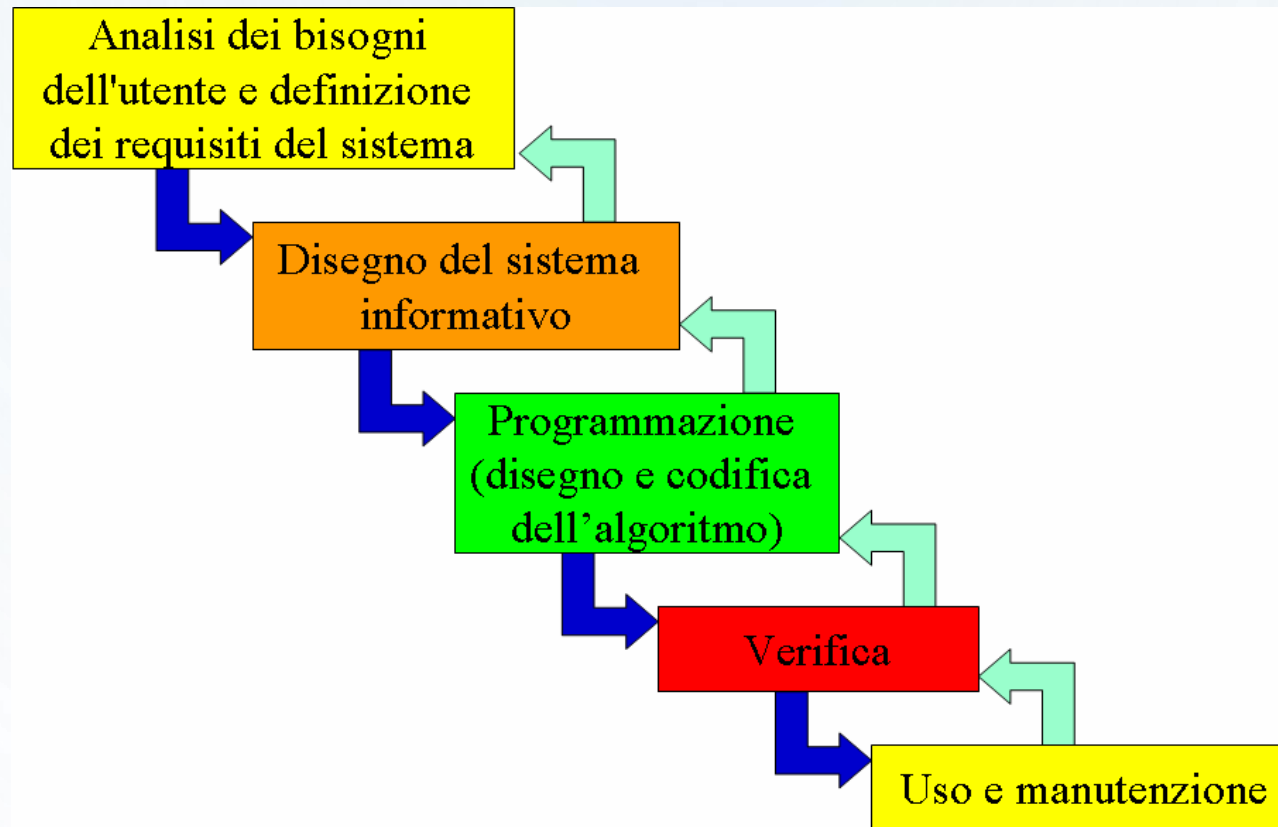
# CICLO DI VITA DEL SOFTWARE

- Il processo di realizzazione di prodotti software viene tipicamente scomposto in sotto attività coordinate tra loro



# CICLO DI VITA DEL SOFTWARE

- Ciclo di vita



# SOFTWARE ENGINEERING

- *L'Ingegneria del Software* è la branca dell'Ingegneria Informatica che raccoglie metodi e tecniche per la produzione del software
  - con fissati parametri di qualità (**requisiti**)
  - in maniera **standard**

# SEPARAZIONE TRA ANALISI E PROGETTO

- Separazione netta, in fase progettuale, tra
  - “cosa” → **analisi dei requisiti e specifiche funzionali**
  - “come” → progetto a diversi livelli di dettaglio
- Vantaggi:
  - Rende indipendente l'analisi dei requisiti da scelte anticipate di progetto
  - Consente di indirizzare le scelte della fase di progetto

# ANALISI DEI REQUISITI

- Acquisisce le informazioni necessarie a comprendere il problema
  - dai colloqui con gli utenti di ogni livello
  - da un esame dell'ambiente in cui il programma sarà utilizzato
- Consente una rappresentazione coerente e completa del problema
  - **Requisiti funzionali**
    - Cosa deve fare il programma e su quali dati deve operare
  - **Requisiti non funzionali**
    - Quali prestazioni il programma deve offrire


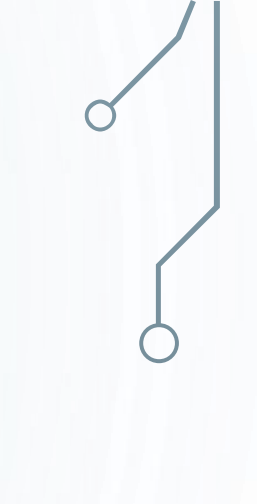
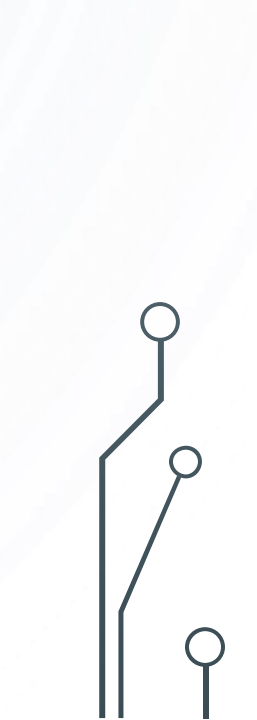


# VERIFICA DELLA CORRETTEZZA DELLE SPECIFICHE

- Le specifiche ottenute nella fase di analisi
  - Devono avere un'unica interpretazione nel contesto delle applicazioni che si descrivono
  - Devono coprire tutte le possibili situazioni coinvolte nel problema
  - La correttezza delle specifiche deve poter essere verificata in qualunque momento, tramite la documentazione su:
    - la definizione dei dati di ingresso al problema
    - la definizione dei dati in uscita dal problema
    - la descrizione di un metodo risolutivo che sulla carta risolva le specifiche del problema



# PROGETTO

- Si articola nelle attività di:
    - raffinamento successivo dei dati
    - disegno e codifica dell'algoritmo
      - esaminare il problema assegnato
      - costruirne un'astrazione
      - tramite un approccio top-down ridurre la complessità del problema
      - rappresentare la risoluzione in maniera formale in un linguaggio di programmazione
- 
- 
- 

# LA PROGRAMMAZIONE STRUTTURATA

- Programmi di buona qualità devono avere le seguenti caratteristiche
  - Leggibilità
  - Documentabilità
  - Modificabilità
  - Provabilità
- La programmazione strutturata consente di ottenere programmi con tali caratteristiche attraverso:
  - la **documentazione**
  - la **modularità**
  - l'uso di **strutture di controllo** ad un ingresso e ad una uscita
  - l'applicazione del metodo **top down** o di quello **bottom up** nella fase di progettazione
  - l'analisi critica del prodotto

# LA PROGRAMMAZIONE STRUTTURATA

- La programmazione strutturata consente di ottenere programmi con tali caratteristiche attraverso:
  - la **documentazione**
  - la **modularità**
  - l'uso di **strutture di controllo** ad un ingresso e ad una uscita
  - l'applicazione del metodo **top down** o di quello **bottom up** nella fase di progettazione
  - l'analisi critica del prodotto

# LA DOCUMENTAZIONE DEI PROGRAMMI

- Strumento fondamentale per la chiarezza e la leggibilità dei programmi
- Consente
  - maggiore **comprensione** del problema che il programma risolve e quindi della correttezza del programma stesso
  - una più semplice **prosecuzione** del progetto ogni qualvolta lo si sia interrotto
  - una più elementare **comunicazione** delle scelte di progetto
  - una più semplice **modificabilità** del programma al variare delle specifiche del problema

# REGOLE DELLA DOCUMENTAZIONE

- Produrre la documentazione **nel corso** del progetto
  - efficacia e completezza della documentazione soltanto se si documentano le scelte nel momento in cui esse vengono fatte
- Inserire la documentazione quanto più possibile **all'interno** del programma
  - migliore gestione ed aggiornamento
- Va articolata in due livelli
  - Documentazione esterna del programma
  - Documentazione interna del programma

# DOCUMENTAZIONE ESTERNA

- È il primo livello di documentazione e va compilato **preliminarmente** nella fase di analisi dei requisiti
- Descrive soltanto **cosa** fa il programma e non *come* lo fa
- Deve segnalare dettagli operativi tali da rendere **autonomo** l'utente del programma nell'uso dello stesso
  - Funzionalità esterne necessarie
  - Attivazione del programma
  - Diagnostiche di errore
  - Configurazione richiesta del sistema
  - Indicazione sull'installazione del programma
  - Versione e data di aggiornamento

# DOCUMENTAZIONE INTERNA

- Descrive la **struttura interna** del programma in termini di scelte sulle strutture dati e sull'algoritmo
  - Evidenziazione della struttura del programma mediante l'uso dell'**indentazione**
  - Documentazione top down su **come** il programma è stato generato attraverso i vari raffinamenti
  - Uso di **nomi** di variabili autoesplicativi
  - Commento del programma attraverso le **frasi di commento** di cui tutti i linguaggi di programmazione sono dotati
    - Motivazioni
    - Asserzioni



# MOTIVAZIONI

- Descrivono il **significato** di ciascun frammento di programma che realizzi una funzionalità significativa indipendentemente da come la implementi
- Si antepone alla frase di commento la sigla “M:”
  - */\* M: calcolo la somma degli n numeri \*/*
  - */\* M: inverti la matrice \*/*
- Particolarmente importante è il commento di motivazione globale posto all’inizio del programma
  - Descrive in modo succinto il problema che il programma risolve

# ASSERZIONI

- Definisce lo **stato di una o più variabili** a seguito dell'esecuzione delle istruzioni precedenti
- Si antepone alla frase di commento la sigla "A:"
  - */\* A:  $x=2y+c-1$  con  $y>c$  \*/*
- Possono essere utilizzate per effettuare una prova qualitativa della correttezza del programma
- Tra tutte le asserzioni particolare cura va posta su quelle riguardanti le variabili di input
  - Specificano le condizioni limite nelle quali la soluzione adottata si troverà a lavorare

# LA PROGRAMMAZIONE STRUTTURATA

- La programmazione strutturata consente di ottenere programmi con tali caratteristiche attraverso:
  - la **documentazione**
  - la **modularità**
  - l'uso di **strutture di controllo** ad un ingresso e ad una uscita
  - l'applicazione del metodo **top down** o di quello **bottom up** nella fase di progettazione
  - l'analisi critica del prodotto

# MODULARITÀ

- Un programma deve essere composto di **moduli funzionali**
  - Ogni modulo funzionale deve possedere un singolo e ben precisato compito
- Valido supporto per la fase di progetto
  - Rispecchia la necessità di esaminare un solo aspetto di un problema alla volta

# LA PROGRAMMAZIONE STRUTTURATA

- La programmazione strutturata consente di ottenere programmi con tali caratteristiche attraverso:
  - la **documentazione**
  - la **modularità**
  - l'uso di **strutture di controllo** ad un ingresso e ad una uscita
  - l'applicazione del metodo **top down** o di quello **bottom up** nella fase di progettazione
  - l'analisi critica del prodotto

# STRUTTURE DI CONTROLLO AD UN INGRESSO E AD UNA USCITA

- Le strutture di controllo sono da considerarsi degli **schemi di composizione dei moduli** costituenti il programma
- Definiscono il controllo del flusso di esecuzione di un programma, ovvero servono a specificare se, quando, in quale ordine e quante volte devono essere eseguite le istruzioni di un programma
- Devono assolutamente avere un solo ingresso ed una sola uscita
- Perché l'integrazione di moduli tramite le strutture di controllo deve generare un modulo risultante con un solo punto di ingresso ed un solo punto di uscita

# LA PROGRAMMAZIONE STRUTTURATA

- La programmazione strutturata consente di ottenere programmi con tali caratteristiche attraverso:
  - la **documentazione**
  - la **modularità**
  - l'uso di **strutture di controllo** ad un ingresso e ad una uscita
  - l'applicazione del metodo **top down** o di quello **bottom up** nella fase di progettazione
  - l'analisi critica del prodotto

# TOP DOWN E STEPWISE REFINEMENT

- Costituisce il **modo procedurale** di raggiungimento della soluzione (metodo deduttivo)
  - Si procede dal generale al particolare per “raffinamenti successivi”
- Si analizza il problema al più alto livello possibile di astrazione individuandone gli elementi più importanti
  - Si suppone l'esistenza di un sistema adatto ad eseguire tali elementi
- Ogni elemento, a sua volta, diventa il problema da analizzare
  - Suddivisione in problemi più elementari
- Il procedimento continua fino a raggiungere
  - un livello di scomposizione comprensibile all'esecutore
  - un software in uso



# PROCESSO DI RAFFINAMENTO PER LIVELLI DI ATRAZIONE

- L'**astrazione** consiste nell'estrazione di dettagli essenziali mentre vengono omessi i dettagli non essenziali
  - Ogni livello presenta una visione astratta dei livelli più bassi
  - I moduli di livello più alto specificano gli obiettivi di qualche azione oppure cosa deve essere fatto
  - I moduli di livello più basso descrivono come l'obiettivo verrà raggiunto
- Iterativamente si producono **decomposizioni** dell'algoritmo sempre più raffinate e vicine alla sua espressione finale nel linguaggio di programmazione

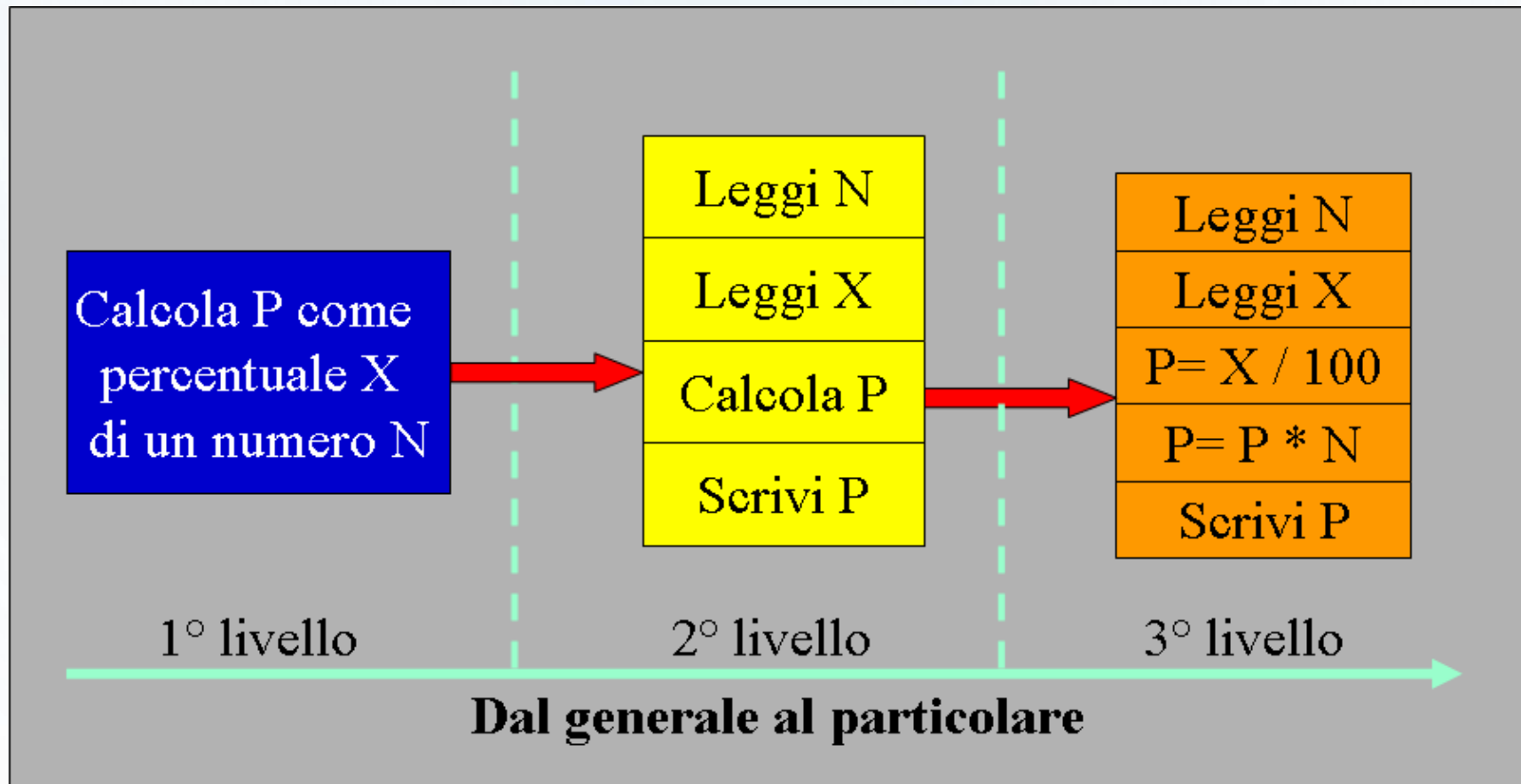
# PROCESSO DI RAFFINAMENTO PER LIVELLI DI ASTRAZIONE

- Ogni sottoproblema che si individua può essere espresso con
  - una frase del linguaggio naturale sintetica ed espressiva
  - con un insieme di istruzioni del linguaggio di programmazione
    - se la funzionalità è abbastanza elementare
- Nel primo caso il sottoproblema diventa un nuovo problema da affrontare
  - Devono essere definite e documentate le variabili di ingresso e di uscita e le interfacce tra i vari sottoproblemi

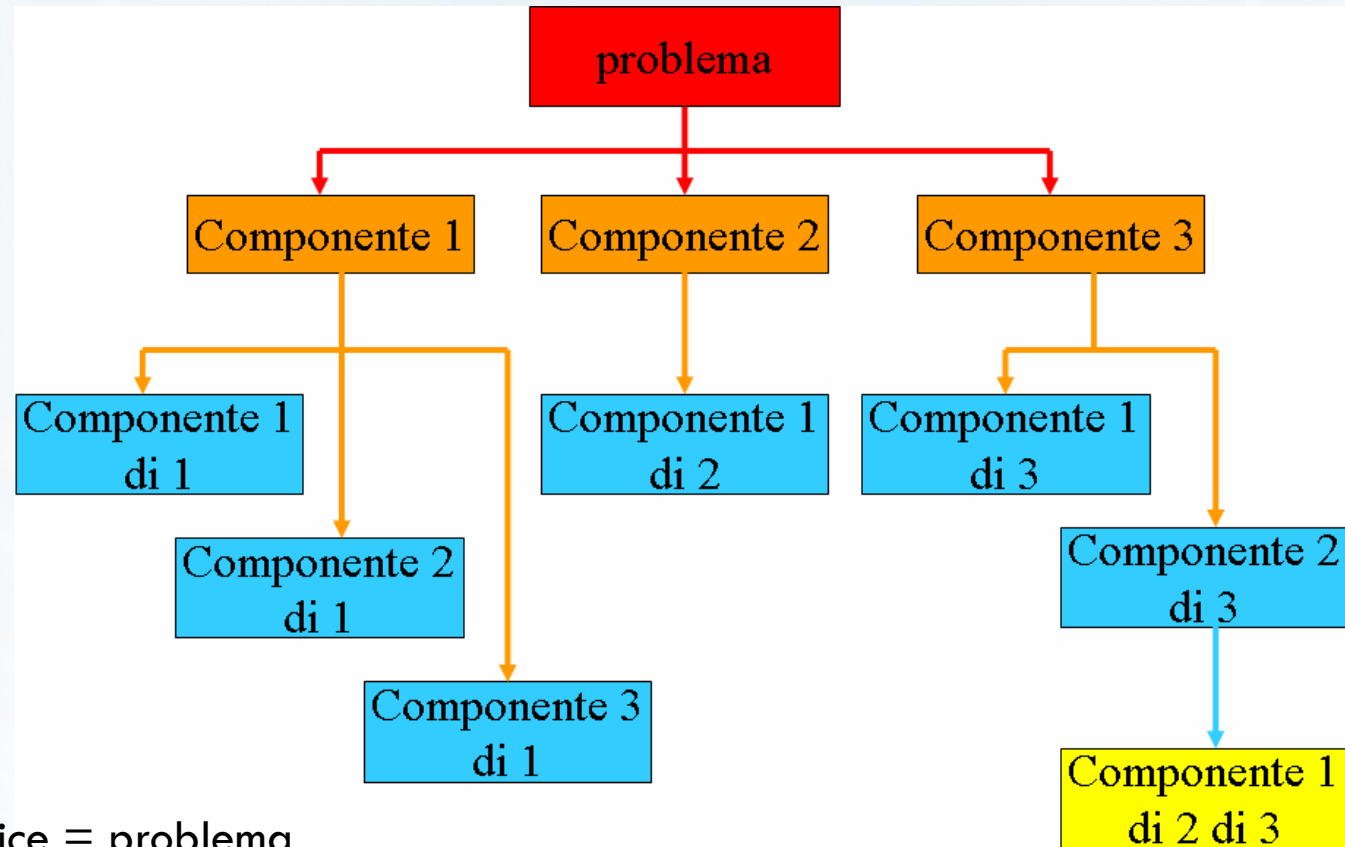
# PROCESSO DI RAFFINAMENTO PER LIVELLI DI ASTRAZIONE

- Ad ogni passo della decomposizione si devono organizzare i sottoproblemi in:
  - Sequenza
    - quando dall'analisi del problema ci si accorge che le attività devono essere svolte una di seguito all'altra
  - Alternativa
    - quando si deve scegliere tra una o più attività
  - Iterativa
    - quando una o più attività devono essere eseguite più volte

# ESEMPIO TOP-DOWN: CALCOLO DI UNA PERCENTUALE



# SOLUZIONE DEL PROBLEMA: ALBERO

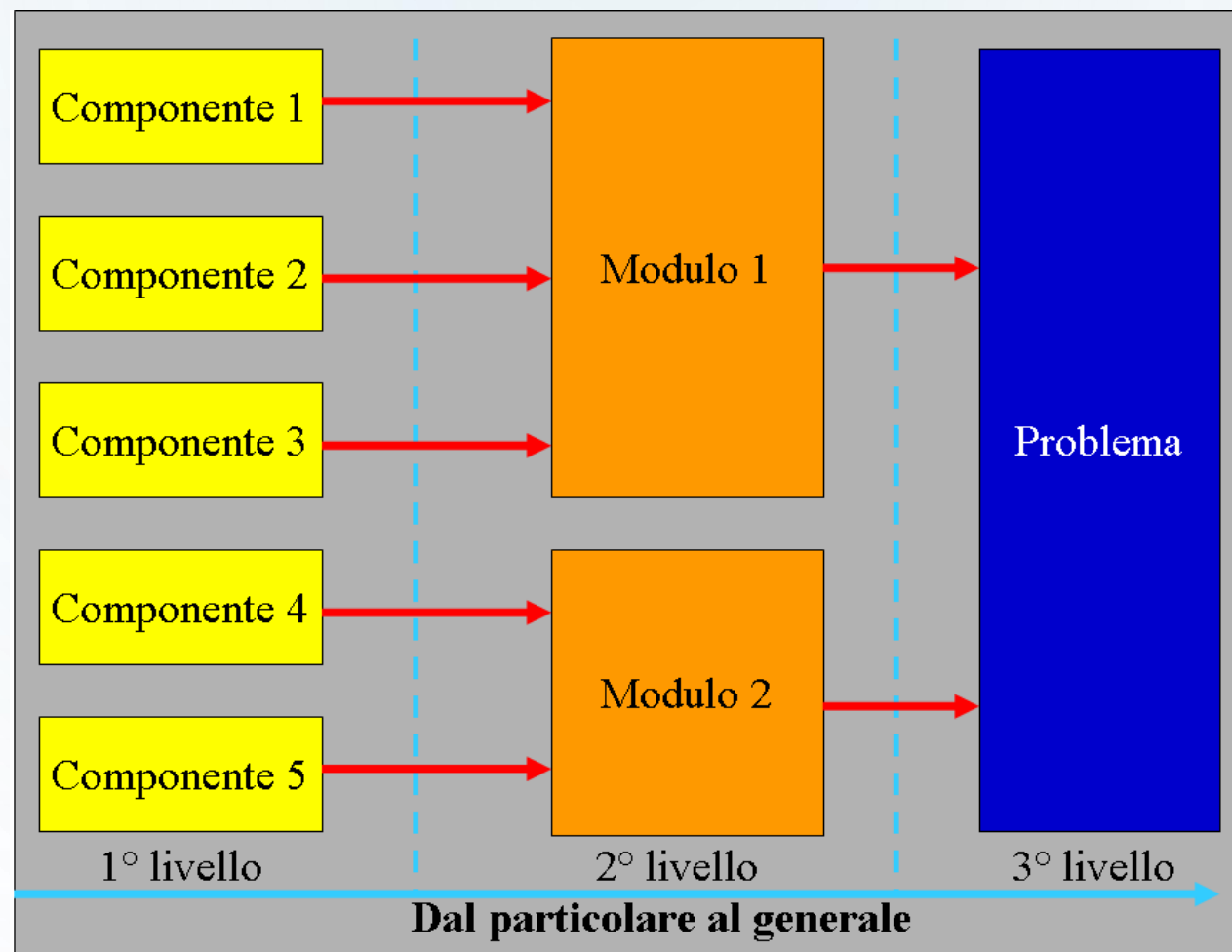


- Radice = problema
- Nodi = decisioni di progetto
- Foglie = descrizione della soluzione in modo comprensibile all'esecutore

# BOTTOM UP

- Considera il sistema reale a disposizione (metodo induttivo)
- Si creano progressivamente moduli elementari che opportunamente integrati formano moduli capaci di compiere operazioni più complesse
- Il procedimento continua fino a quando è stato creato un modulo che corrisponde alla soluzione del problema
- Nella struttura ad albero si procede dalle foglie verso la radice

# BOTTOM UP



# LA PROGRAMMAZIONE STRUTTURATA

- La programmazione strutturata consente di ottenere programmi con tali caratteristiche attraverso:
  - la **documentazione**
  - la **modularità**
  - l'uso di **strutture di controllo** ad un ingresso e ad una uscita
  - l'applicazione del metodo **top down** o di quello **bottom up** nella fase di progettazione
  - **l'analisi critica del prodotto**



# ANALISI CRITICA

- Effettua una minuziosa valutazione della soluzione adottata
- Verifica la correttezza della versione dell'algoritmo
  - mediante ad esempio una simulazione della sua esecuzione fissando un set di dati in ingresso
- Valuta l'efficienza della soluzione adottata
  - confronto con altre soluzioni
  - studio dell'impatto di una particolare scelta di progetto
- Verifica che ogni azione sia documentata
  - unitamente alle specifiche iniziali si deve avere a disposizione una documentazione utilizzabile per una futura ristrutturazione dell'algoritmo

# ELEMENTI LESSICALI

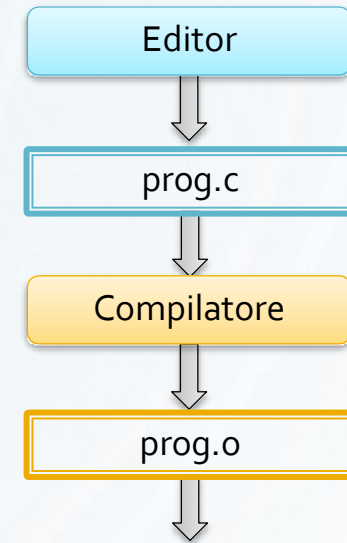
- Un programma, scritto in un qualsiasi linguaggio di programmazione, prima di essere eseguito viene sottoposto ad un processo di traduzione
- Lo scopo di questo processo è quello di tradurre il programma originale (codice sorgente) in uno semanticamente equivalente, ma eseguibile su una certa macchina
- Il processo di traduzione è suddiviso in più fasi, ciascuna delle quali volta all'acquisizione di opportune informazioni necessarie alla fase successiva

# ELEMENTI LESSICALI

- La prima di queste fasi è nota come analisi lessicale ed ha il compito di riconoscere gli elementi costitutivi del linguaggio sorgente, individuandone anche la categoria lessicale
- Ogni linguaggio prevede un certo numero di categorie lessicali e in C/C++ possiamo distinguere in particolare le seguenti categorie:
  - Commenti
  - Identificatori
  - Parole riservate
  - Costanti letterali
  - Segni di punteggiatura e operatori

# TRADUZIONE DEI PROGRAMMI

- Per essere eseguito da una CPU, un algoritmo deve essere espresso in linguaggio macchina
  - insieme di istruzioni che la CPU comprende
    - sequenze di bit
- C e C++ sono linguaggi *di alto livello*
  - la loro potenza espressiva è superiore a quella del linguaggio macchina
  - sono stati pensati per aiutare i programmatori
- Per tutti i linguaggi di programmazione esiste
  - una **grammatica**
  - il programma di **traduzione** in linguaggio macchina



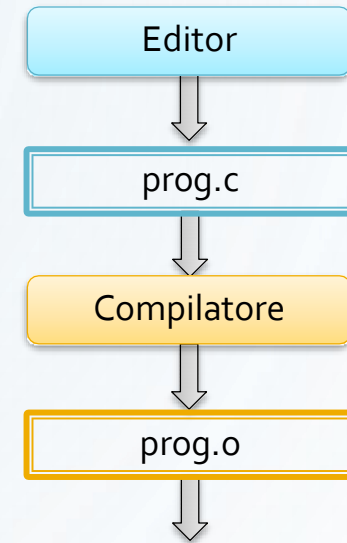
# TRADUZIONE DEI PROGRAMMI

## ■ Compilazione

- la traduzione avviene una volta sola
  - velocità di esecuzione
  - il programma dipende strettamente dalla CPU per la quale è stato prodotto
- ad esempio, Fortran e C/C++

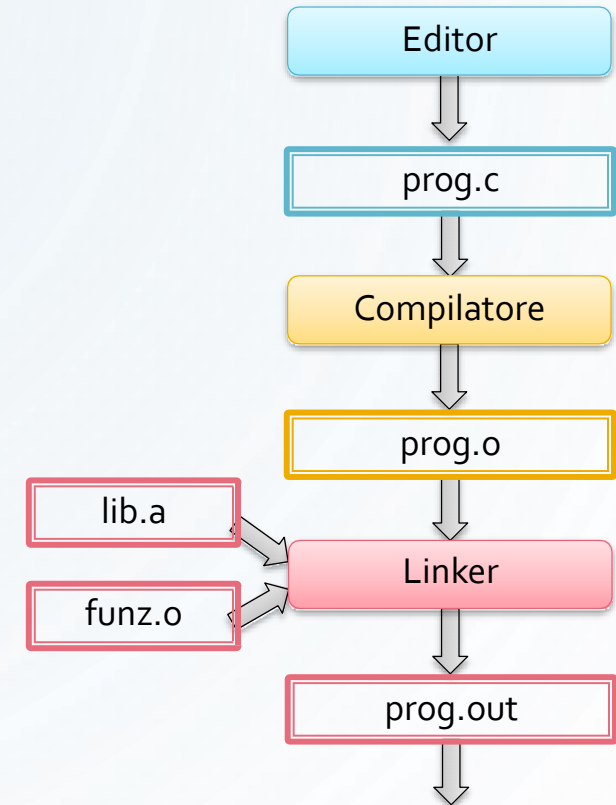
## ■ Interpretazione

- la traduzione avviene ogni volta che il programma viene eseguito
  - tempi più lunghi
  - maggiore portabilità
- ad esempio, Python e PHP



# COMPILAZIONE DEI PROGRAMMI

- Nell'approccio per compilazione la sola traduzione non è sufficiente a rendere il programma «eseguibile» dalla CPU
  - servono delle funzionalità raccolte in librerie
    - interazione con il sistema operativo
    - gestione dell'I/O
- **Linker**
  - assembla tutti gli oggetti e librerie necessari per generare un programma che sia eseguibile dalla CPU



# COMPILAZIONE DEI PROGRAMMI

## ■ Loader o Caricatore

- carica in memoria il programma
- attiva il programma

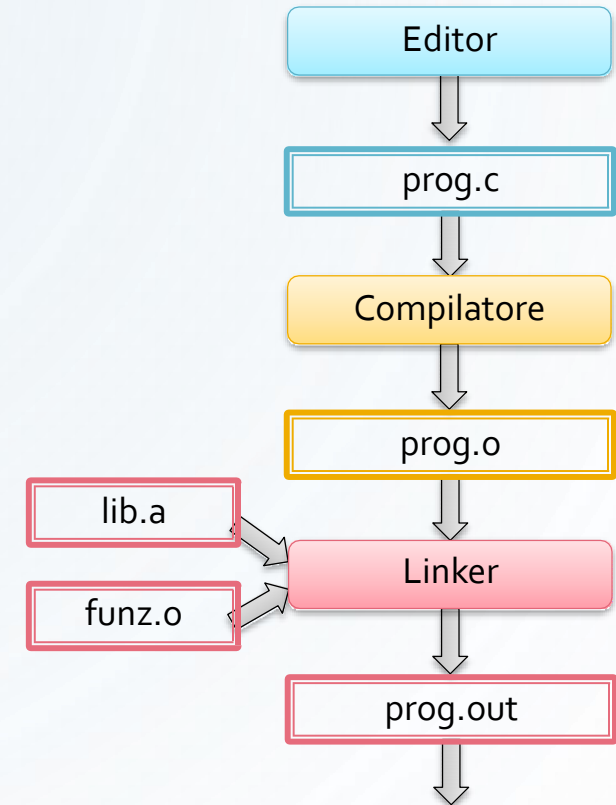
## ■ Il suffisso dei file dipende dal sistema operativo

### ■ *Windows*

- Sorgente: .c / .cpp
- Oggetto: .obj
- Libreria: .lib
- Eseguitibile: .exe

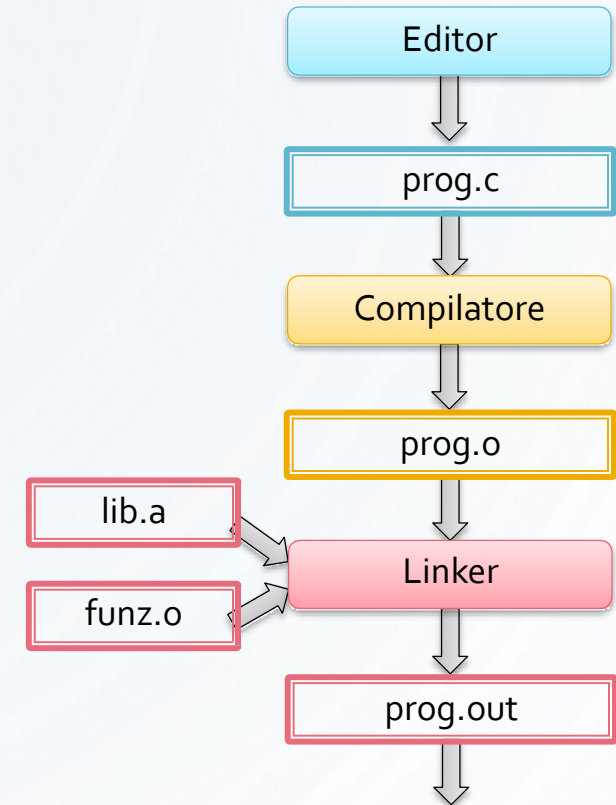
### ■ *Unix*

- Sorgente: .c / .cpp
- Oggetto: .o
- Libreria: .a
- Eseguitibile: .out



# COMPILAZIONE DEI PROGRAMMI

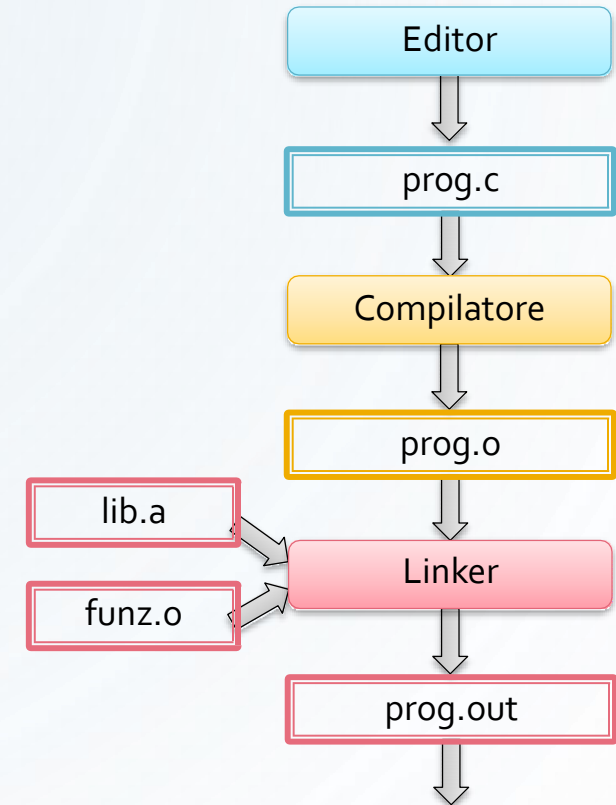
- Ambienti di sviluppo integrato (IDE)
  - in essi sono presenti funzionalità per
    - scrivere e modificare codice sorgente (editor)
    - tradurre il codice sorgente (compilatore)
    - generare l'eseguibile (linker)
    - effettuare il debugging
  - tipicamente fanno da front-end verso compilatori a linea di comando
    - ad esempio *gcc* e *g++*, che sono pienamente compatibili con gli standard ANSIC e ANSIC++





# COMPILAZIONE DEI PROGRAMMI

- Ambienti di sviluppo integrato (IDE)
  - in essi sono presenti funzionalità per
    - scrivere e modificare codice sorgente (editor)
    - tradurre il codice sorgente (compilatore)
    - generare l'eseguibile (linker)
    - effettuare il debugging
  - tipicamente fanno da front-end verso compilatori a linea di comando
    - ad esempio *gcc* e *g++*, che sono pienamente compatibili con gli standard ANSIC e ANSIC++



# ERRORI NEI PROGRAMMI

## ■ Classificazione

### ■ Errori che si verificano

- Nella compilazione del sorgente
- Nel collegamento di oggetti e librerie
- Nel caricamento dell'eseguibile in memoria
- Nell'esecuzione

} Il programma non arriva alla fase di esecuzione

└─ Se non gestiti Il programma può terminare improvvisamente

# ERRORI NEI PROGRAMMI

## ■ Errori più frequenti

### ■ Nella compilazione

- Assenza del ";" per chiudere le istruzioni
- Mancanza di parentesi () per una funzione senza parametri
- Errori dovuti al Case Sensitive

### ■ Nel collegamento

- Uso di un elemento in un sorgente diverso dal modo in cui è definito in un altro sorgente

### ■ Nel caricamento

- Un programma necessita di più memoria di quella esistente

### ■ Nell'esecuzione (anche chiamati eccezioni)

- Tipicamente sono errori logici
  - Es.: divisione per 0, operazioni ripetute all'infinito (loop)

**DOMANDE, DUBBI, PERPLESSITÀ**

