Università degli Studi di Palermo

Corso di Laurea
Ingegneria Gestionale

Lezione 5

Programmi e Linguaggi di Programmazione

Ing. Massimo Cossentino

Sistemi Informativi Aziendali

a.a. 2008/2009

Cosa è un Programma

- Algoritmo: una successione di operazioni che permette, a partire da dati iniziali, di ottenere la soluzione di un determinato problema.
- Analogo ad una ricetta di cucina con tutte le istruzioni per preparare un certo piatto a partire dalle materie prime.
- Un PROGRAMMA è un algoritmo scritto in un linguaggio che un computer può/sa interpretare ed eseguire

Cosa è un Programma

- Una definizione più formale:
  - La descrizione di un algoritmo per la soluzione di un problema P in un linguaggio di programmazione L, e cioè un programma in L, è costituito da una sequenza finita di istruzioni in L la cui esecuzione da parte del calcolatore porta alla risoluzione di P.

Cos'è un linguaggio

- Definizione 1 – Un linguaggio è un insieme di parole e di metodi di combinazione delle parole usati e compresi da una comunità di persone
  - è una definizione poco precisa perché non evita le ambiguità dei linguaggi naturali, non si presta a descrivere processi computazionali automatici, non aiuta a stabilire proprietà

- Definizione 2 – Il linguaggio è un sistema matematico che consente di rispondere a domande come:
  - quali sono le frasi lecite?
  - si può stabilire se una frase appartiene al linguaggio?
  - come si stabilisce il significato di una frase?
  - quali sono gli elementi linguistici primitivi?

Lessico, sintassi e semantica

- Lessico: l'insieme di regole formali per la scrittura di *parole* in un linguaggio
- Sintassi: l'insieme di regole formali per la scrittura di *frasi* in un linguaggio, che stabiliscono la grammatica del linguaggio stesso
- Semantica: l'insieme dei *significati* da attribuire alle frasi (sintatticamente corrette) costruite nel linguaggio

- N.B: una frase può essere sintatticamente corretta e tuttavia non avere significato!

Esempio: la sintassi di un numero naturale

<cifra-non-nulla> := 1|2|3|4|5|6|7|8|9

<cifra> := 0 | <cifra-non-nulla>

<naturale> := 0 | <cifra-non-nulla>{<cifra>}

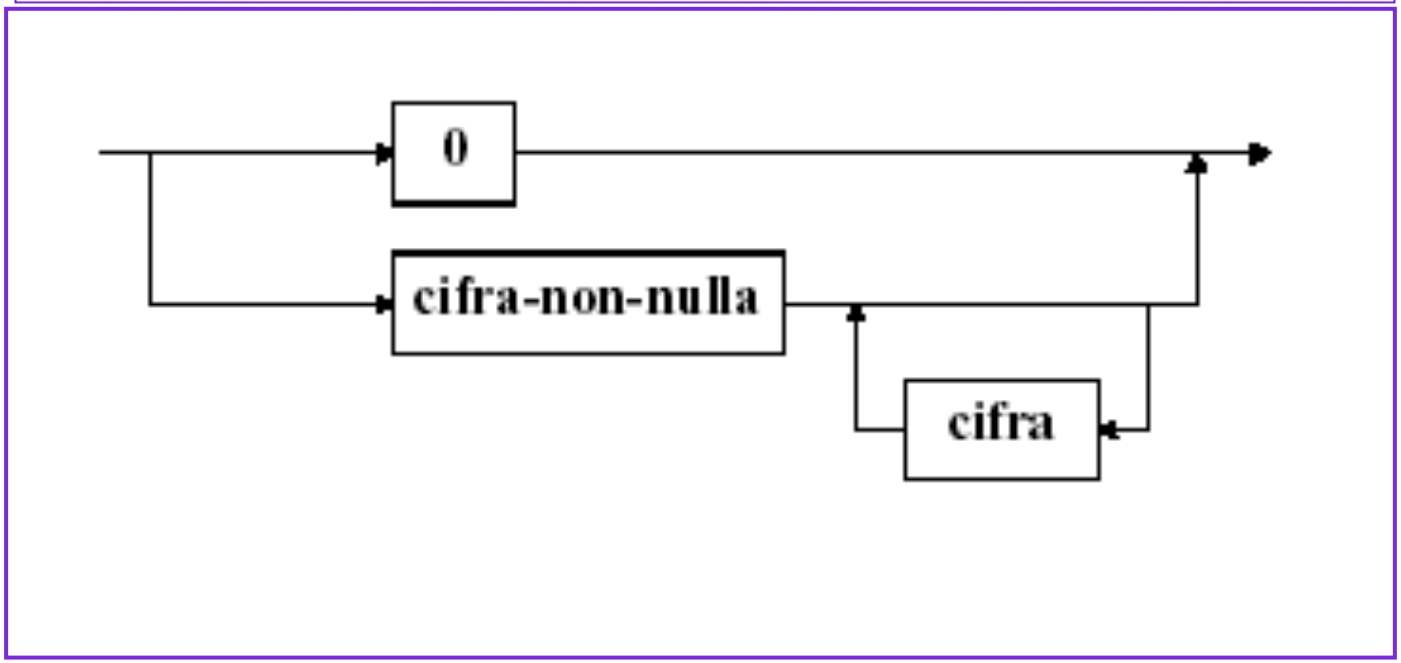


Diagramma sintattico

L'evoluzione dei linguaggi di programmazione

- Benché siano macchine in grado di compiere operazioni apparentemente complesse, i calcolatori devono essere "guidati" per mezzo di istruzioni appartenenti ad un linguaggio, spesso limitato, a loro comprensibile
- Un *linguaggio di programmazione* è costituito, come ogni altro tipo di linguaggio, da un alfabeto di simboli, con cui vengono costruite le parole e le frasi, e da un insieme di regole lessicali e sintattiche per la costruzione e l'uso corretto delle parole e delle frasi del linguaggio
- A basso livello (hardware) i calcolatori riconoscono ed eseguono solo comandi semplici, come *trasferisci un numero*, *addiziona due numeri, confronta due numeri, …*

L'evoluzione dei linguaggi di programmazione

- Tali comandi costituiscono il set di istruzioni macchina e i programmi che li utilizzano direttamente sono i programmi in linguaggio macchina
- In linguaggio macchina
  - l'effettuazione di operazioni anche semplici richiede l'esecuzione di numerose istruzioni
  - qualunque entità (istruzioni, variabili, dati) è rappresentata in binario: i programmi sono difficili da scrivere, leggere e manutenere
- Il linguaggio macchina riflette l'organizzazione della macchina più che la natura del problema da risolvere

L'evoluzione dei linguaggi di programmazione

- Nei primi anni '50, una prima evoluzione, rispetto alla programmazione in linguaggio macchina, fu rappresentata dall'introduzione dei linguaggi di assemblatore
- In linguaggio di assemblatore
  - ogni istruzione è identificata da un codice simbolico *mnemonico* piuttosto che da un codice numerico; tali mnemonici costituiscono altrettante *parole chiave* del linguaggio
  - il riferimento alle locazioni di memoria (e ai registri) viene effettuato per mezzo di *nomi simbolici* rappresentativi dei corrispondenti indirizzi: nascono le *variabili*
  - anche le modalità di indirizzamento vengono rappresentate in modo simbolico

Esempio di linguaggio assembler

0	0100000000010000	leggi il primo valore (*a*, va nella cella 16)	read	*primo*	
1	0100000000010001	leggi il secondo valore (*b*, cella 17)		read	*secondo*
2	0100000000010010 *terzo*	leggi il terzo valore (*c*, cella 18)			read
3	0100000000010011 *quarto*	leggi il quarto valore (*d*, cella 19)			read
4	0000000000010000	carica A con il contenuto della cella 16		loada	*primo*
5	0001000000010001	carica B con il contenuto della cella 17		loadb	*secondo*
6	0110000000000000	somma i due valori (il risultato è in A)		add	
7	0010000000010100	salva il risultato parziale nella cella 20		storea	*risultato*
8	0000000000010010	carica A con il contenuto della cella 18		loada	*terzo*
9	0001000000010011	carica B con il contenuto della cella 19		loadb	*quarto*
10	0110000000000000	somma i due valori (il risultato è in A)		add	
11	0001000000010100	carica B con il contenuto della cella 20		loadb	*risultato*
12	1000000000000000	moltiplica i due valori (il risultato è in A)	mul		
13	0010000000010100 storea *risultato*	salva il risultato nella cella 20			
14	0101000000010100 *risultato*	scrivi il contenuto della cella 20			write
15	1101000000000000 *halt*	arresta l'esecuzione			
16		*primo*	cella riservata ad *a*		
17		*secondo*	cella riservata a *b*		
18		*terzo*	cella riservata a *c*		
19		*quarto*	cella riservata a *d*		
20	*risultato*		cella riservata al risultato parziale		
.....					
.....		celle libere			
.....					

L'evoluzione dei linguaggi di programmazione

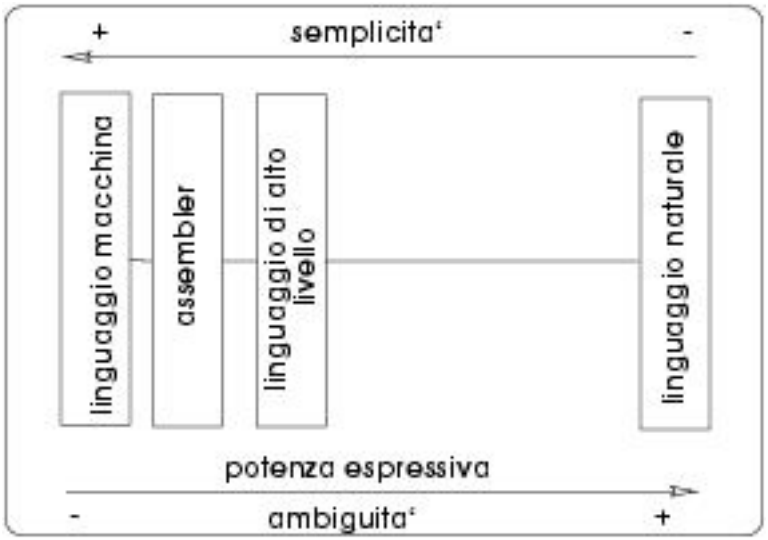
- Naturalmente, per essere eseguiti i programmi scritti in linguaggio di assemblatore devono essere tradotti in linguaggio macchina: alle parole chiave deve essere sostituito il codice operativo, agli indirizzi simbolici gli indirizzi reali
- Questo compito, ripetitivo e noioso ma semplice, può essere affidato alla macchina stessa, guidata da un apposito programma, chiamato assemblatore
- Il vantaggio fondamentale derivante dall'uso del linguaggio di assemblatore è evidente: il programmatore non deve ricordare sequenze astruse di numeri binari

L'evoluzione dei linguaggi di programmazione

- I difetti dei linguaggi di assemblatore si possono così riassumere:
  - sono comunque legati all'architettura della macchina
  - sono poco espressivi
  - sono inadeguati a gestire l'enorme complessità dei programmi moderni
- Oggi si utilizzano i linguaggi di assemblatore solo per la scrittura di programmi o porzioni di programmi caratterizzati da vincoli molto stringenti sui tempi di esecuzione

L'evoluzione dei linguaggi di programmazione

- Di norma, si usano linguaggi più vicini al linguaggio naturale, detti linguaggi di alto livello
- I linguaggi di alto livello possono essere considerati elementi intermedi di una scala di linguaggi ai cui estremi si trovano il linguaggio macchina, al gradino più basso, ed i linguaggi naturali, come l'italiano e l'inglese, al gradino più alto



- I linguaggi di programmazione differiscono comunque dai linguaggi naturali: sono infatti meno *espressivi* ma più *precisi*
- Sono *semplici* e *poveri* (poche parole, poche regole), ma *non ambigui*

Astrazione

- Si parla di programmazione di basso livello quando si utilizza un linguaggio molto vicino alla macchina
- Si parla invece di programmazione di alto livello quando si utilizzano linguaggi più sofisticati ed astratti, slegati dal funzionamento fisico della macchina
- Si viene così a creare una gerarchia di linguaggi, dai meno evoluti (il linguaggio macchina o l'assembler) ai più evoluti (*Pascal, Perl, Java, etc.*)
- In questa gerarchia il linguaggio *C* viene posto da alcuni ad un livello inferiore rispetto a quello dei linguaggi più evoluti

Astrazione

- Esistono, quindi, diversi livelli di astrazione:
- Linguaggi macchina e assemblatore
  - Implicano la conoscenza dettagliata delle caratteristiche della macchina (registri, dimensione dati, set di istruzioni)
  - Semplici algoritmi implicano la specifica di molte istruzioni
- Linguaggi di alto livello
  - Il programmatore può trascurare i dettagli legati all'architettura ed esprimere i propri algoritmi in modo simbolico
  - Sono indipendenti dalla macchina sottostante

Linguaggi di programmazione di alto livello

- Richiedono un compilatore o un interprete in grado di *tradurre* le istruzioni del linguaggio di alto livello in istruzioni macchina di basso livello, eseguibili dal calcolatore
- Un compilatore è un programma traduttore simile ad un assemblatore, ma più complesso, infatti
  - esiste una corrispondenza biunivoca fra istruzioni in assembler ed istruzioni macchina
  - ogni singola istruzione di un linguaggio di alto livello corrisponde di norma a molte istruzioni in linguaggio macchina: quanto più il linguaggio si discosta dal linguaggio macchina, tanto più il lavoro di traduzione del compilatore risulta complesso

Linguaggi di programmazione di alto livello

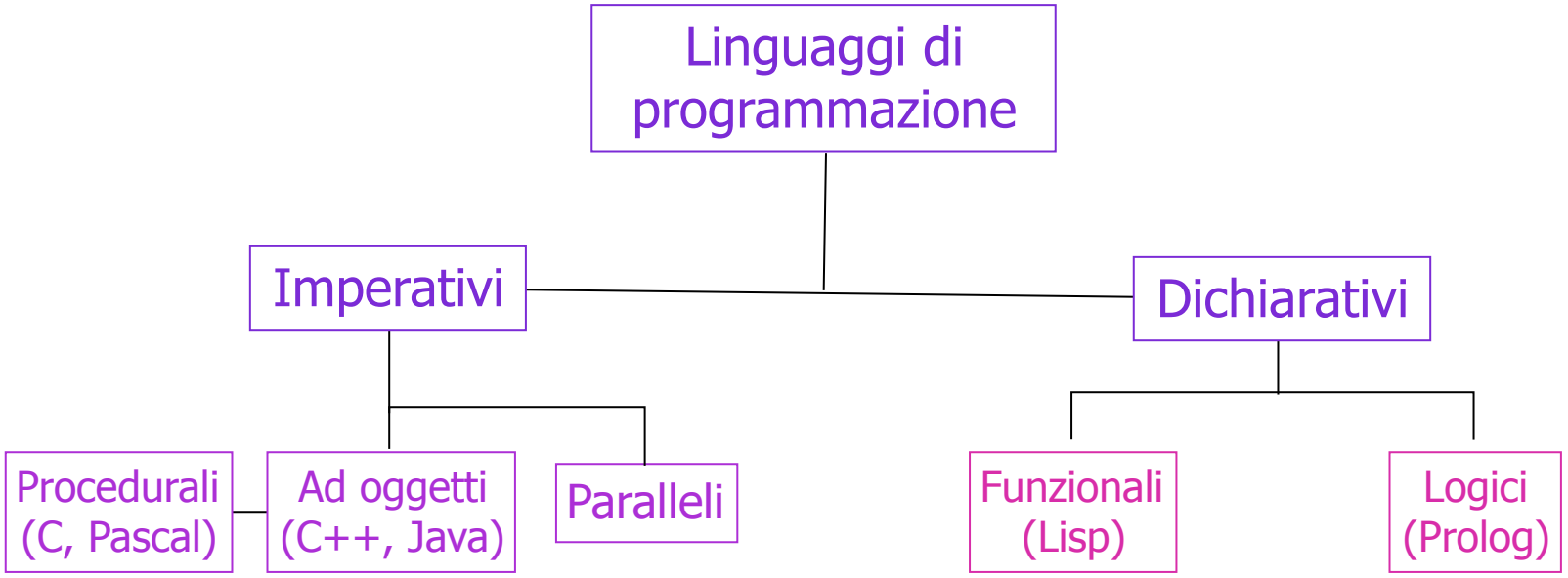
- I linguaggi che non dipendono dall'architettura della macchina offrono alcuni vantaggi fondamentali:
  - il programmatore non deve conoscere i dettagli architetturali di ogni calcolatore;
  - il programmatore può prescindere da dettagli inessenziali ai fini della soluzione di un problema, e concentrarsi così sugli aspetti fondamentali;
  - i programmi scritti per un calcolatore possono essere utilizzati su qualsiasi altro calcolatore, previa ricompilazione (portabilità);
  - la (relativa) prossimità con i linguaggi naturali rende i programmi più semplici, non solo da scrivere, ma anche da leggere (leggibilità);
  - la possibilità di codificare algoritmi in maniera astratta si traduce in una migliore comprensibilità del codice e quindi in una più facile analisi di correttezza;
  - è più semplice apportare ai programmi modifiche di tipo correttivo, perfettivo, evolutivo e adattivo (manutenibilità)

Linguaggi di programmazione di alto livello

- Un possibile svantaggio dell'uso dei linguaggi di alto livello è la riduzione di efficienza:
  - È infatti possibile utilizzare istruzioni macchina diverse per scrivere programmi funzionalmente equivalenti: il programmatore ha un controllo limitato sulle modalità con cui il compilatore traduce il codice
  - Tuttavia i moderni compilatori sono spesso *ottimizzati*, in grado cioè di "scegliere" soluzioni efficienti in modo trasparente per il programmatore
- Poichè la maggior parte dei costi di produzione del software è localizzata nella fase di manutenzione, per la quale leggibilità e portabilità sono cruciali, l'uso dei linguaggi evoluti è imprescindibile

Linguaggi di programmazione di alto livello

- I numerosi linguaggi di programmazione esistenti possono essere classificati sulla base del *modello astratto di programmazione* che sottintendono e che è necessario adottare per utilizzarli:



Linguaggi di programmazione di alto livello

- Linguaggi imperativi
  - Il modello computazionale è basato sul *cambiamento di stato della memoria* della macchina
  - È centrale il concetto di *assegnazione di un valore* ad una locazione di memoria
  - Il compito del programmatore è costruire una sequenza di assegnazioni che producano lo stato finale (in modo tale che questo rappresenti la soluzione del problema)
- Linguaggi dichiarativi
  - Il modello computazionale è basato sui concetti di *funzione e relazione*
  - Il programmatore non ragiona in termini di assegnazioni di valori, ma di relazioni tra entità e di valori di una funzione

Linguaggi di programmazione di alto livello

- Sulla base dell'ambito in cui si colloca il problema da risolvere, è opportuno adottare un linguaggio piuttosto che un altro:
  - Calcolo scientifico: Fortran, C, Matlab
  - Intelligenza Artificiale: Prolog, Lisp, C
  - Applicazioni gestionali: Cobol, SQL, C
  - Sistemi operativi: Assembler, C, Java
  - Applicazioni visuali: C++, Java, Visual Basic
  - Applicazioni Web: Java, PHP, ASP

Compilatori ed interpreti

- Affinché un programma scritto in un qualsiasi linguaggio di programmazione sia comprensibile (e quindi eseguibile) da parte di un calcolatore, occorre tradurlo dal linguaggio originario al linguaggio della macchina.
- Ogni traduttore è in grado di comprendere e tradurre un solo linguaggio.
- Il traduttore converte il testo di un programma scritto in un particolare linguaggio di programmazione (*sorgente*) nella corrispondente rappresentazione in linguaggio macchina (programma *eseguibile*)

PROGRAMMA	TRADUZIONE
`main()`	
`{ int A;`	`00100101`
`...`	
`A=A+1;`	`11001..`
`if....`	`1011100..`

Compilatori ed interpreti

- Compilatore: opera la traduzione di un programma sorgente (scritto in linguaggio di alto livello) in un programma oggetto direttamente eseguibile dal calcolatore
  - PRIMA si traduce tutto il programma
  - POI si esegue la versione tradotta.
- Interprete: traduce ed esegue il programma sorgente istruzione per istruzione
  - Traduzione ed esecuzione sono intercalate.

Compilatori ed interpreti

- Esempio di compilatore
  - Dobbiamo sottoporre un curriculum, in inglese, ad una azienda, ma non conosciamo l'inglese
  - Abbiamo bisogno di un traduttore che traduca quanto scritto da noi dall'italiano all'inglese
    - contattiamo il traduttore
    - il traduttore riceve il testo da tradurre
    - il traduttore fornisce il testo tradotto
    - possiamo sottoporre il nostro curriculum all'azienda

Compilatori ed interpreti

- Esempio di interprete
  - Dobbiamo incontrare un manager cinese per motivi di lavoro ma non conosciamo il cinese
  - Abbiamo bisogno di un interprete che traduca il nostro dialogo
    - contattiamo l'interprete
    - parliamo in italiano, in presenza dell'interprete
    - *contemporaneamente* l'interprete comunica al manager cinese quanto detto da noi e viceversa
  - Il compito dell'interprete si svolge contestualmente all'incontro col manager cinese

Compilatori ed interpreti

- Riassumendo…
  - I compilatori traducono un intero programma nel linguaggio macchina del calcolatore in uso:
    - traduzione e esecuzione procedono separatamente
    - al termine della compilazione è disponibile la versione tradotta del programma
    - la versione tradotta è specifica per quella macchina
    - per eseguire il programma basta avere disponibile la versione tradotta (non è necessario ricompilare)
  - Gli interpreti invece traducono e immediatamente eseguono il programma *istruzione per istruzione*:
    - traduzione ed esecuzione procedono insieme
    - al termine non vi è alcuna versione tradotta del programma originale
    - se si vuole rieseguire il programma occorre anche ritradurlo

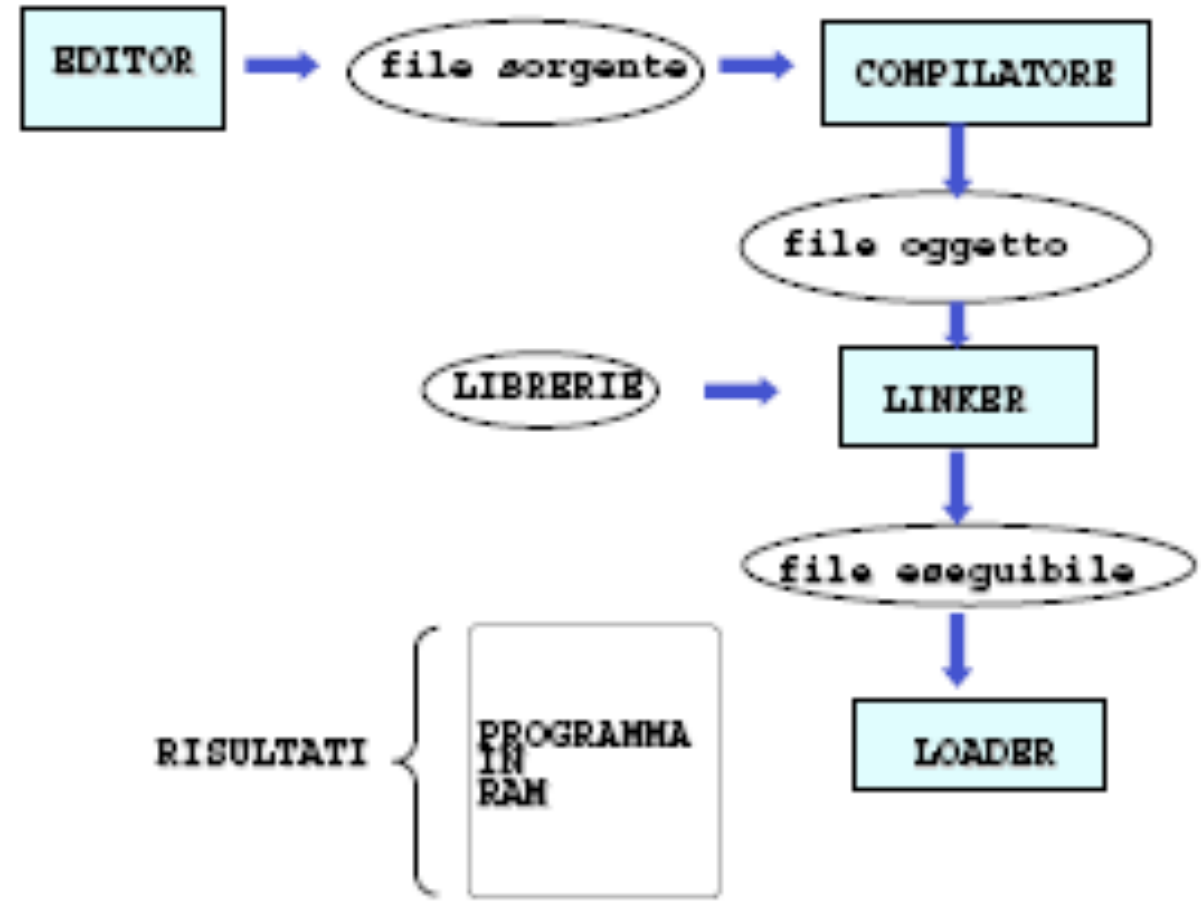
Compilatori ed interpreti

- L'esecuzione di un programma compilato è più veloce dell'esecuzione di un programma interpretato
- I linguaggi interpretati sono tipicamente più flessibili e semplici da utilizzare (nei linguaggi compilati esistono limitazioni alla semantica dei costrutti)
- Per distribuire un programma interpretato si deve necessariamente distribuire il codice sorgente, rendendo possibili operazioni di plagio
- Nei programmi interpretati, è più facile il rilevamento di errori di run–time

Sviluppo di programmi

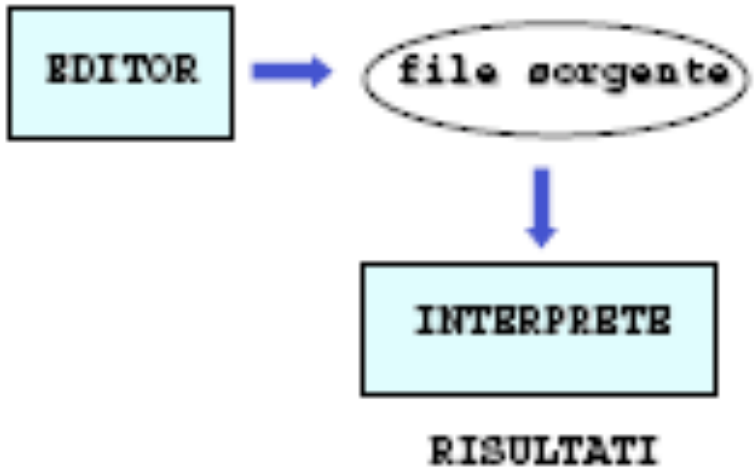
- Si utilizza di norma un *ambiente di programmazione*, cioè un insieme *integrato* di programmi che consentono la scrittura, la traduzione, l'esecuzione, la verifica e la correzione del programma in fase di sviluppo.
- I componenti di un ambiente di programmazione sono:
  - Editor: consente di scrivere il programma *sorgente*, fornendo in uscita un file di testo che lo contiene;
  - Compilatore: effettua la traduzione in linguaggio macchina, fornendo in uscita un file binario contenente il codice *oggetto*;
  - Linker o correlatore: provvede a collegare più moduli oggetto, fornendo in uscita un unico file contenente il programma *eseguibile*;
  - Loader o caricatore: carica in memoria il programma eseguibile e ne avvia l'esecuzione;
  - Debugger: consente l'esecuzione controllata (spesso passo-passo) del programma, al fine di rilevare eventuali errori di funzionamento

Sviluppo di programmi



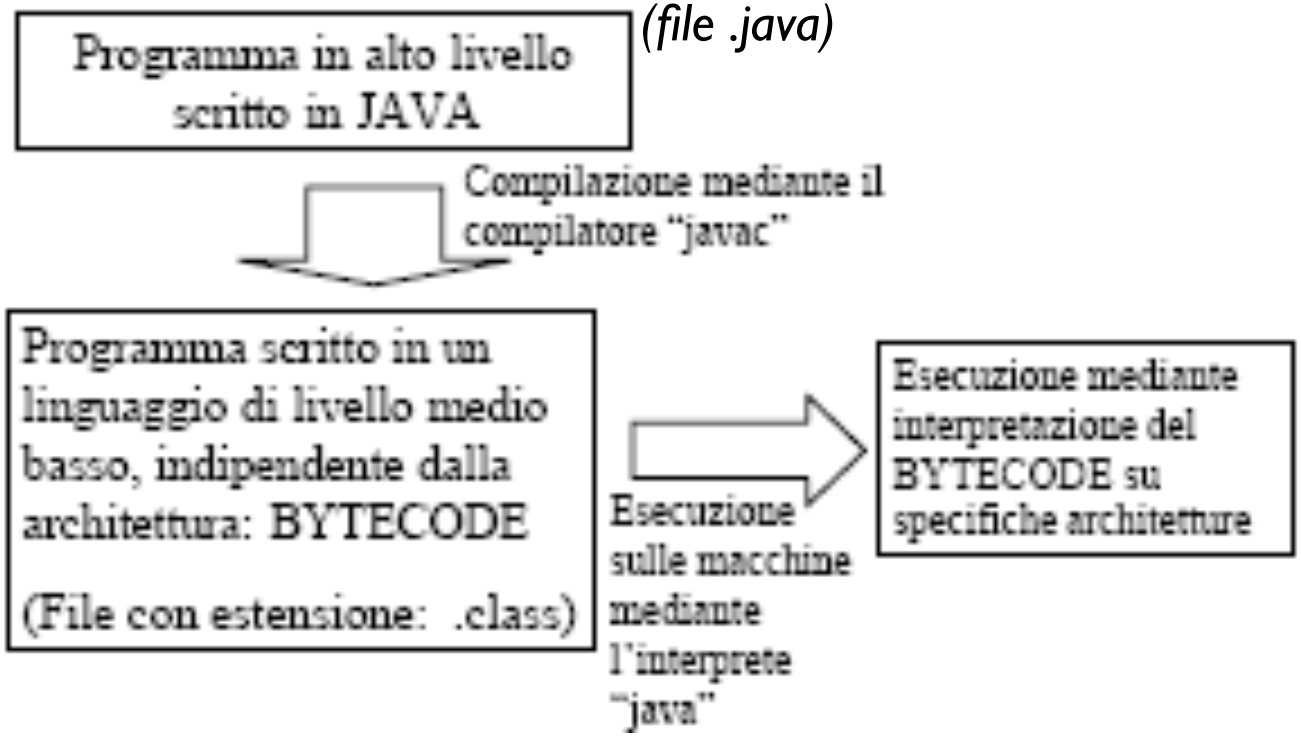
Sviluppo di programmi

- Se si utilizza un interprete lo schema si modifica:



Sviluppo di programmi

- Nel caso di Java:



Sviluppo di programmi

- Il compilatore Java traduce il programma sorgente in una rappresentazione speciale detta *bytecode*
- Il bytecode Java non è un linguaggio macchina di una CPU particolare, ma di una *macchina virtuale Java*
  - L'interprete traduce il bytecode nel linguaggio macchina e lo esegue
- Un compilatore Java non è legato ad una particolare macchina
  - Java è *indipendente* dall'architettura della macchina

Sviluppo di programmi

- Questa soluzione sembra unire i pregi della compilazione:
  - il programma sorgente è tradotto una volta sola e la traduzione da Java a bytecode è relativamente semplice
- con i pregi dell'interpretazione:
  - è facile scrivere un interprete java (Java Virtual Machine) per ogni specifica architettura
- Basta scrivere un solo codice per molte macchine:
  - ideale per il web!
- Il problema dell'efficienza è stato risolto dalle tecniche di compilazione *just-in-time* (JIT)

L'arte della programmazione

- La soluzione di un problema tramite un programma è un procedimento che non si esaurisce nello scrivere codice in un dato linguaggio di programmazione, ma comprende una fase di progetto, che precede, e di verifica, che segue, la scrittura del codice

- Definizione del problema
- Algoritmo per la soluzione del problema

← Analisi

- Codifica
- Debugging
- Validazione
- Documentazione
- Manutenzione

← Programmazione

L'arte della programmazione

- Definizione del problema
  - Definizione degli ingressi e delle uscite
    - quali variabili
    - quale dominio per ogni variabile
  - Risoluzione delle ambiguità
  - Scomposizione in problemi più semplici
- Definizione dell'algoritmo
  - Descrizione verbale
  - Rappresentazione in pseudocodice
  - Rappresentazione mediante diagramma a blocchi strutturato

L'arte della programmazione

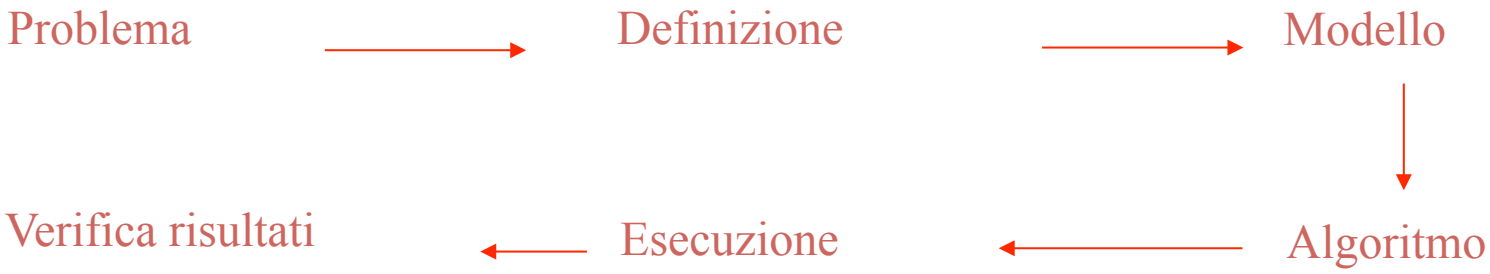
- Codifica
  - Traduzione dell'algoritmo in istruzioni del linguaggio di programmazione
- Debugging, correzione degli errori sintattici e semantici
  - Errori sintattici
    - Espressioni non valide o non ben formate nel linguaggio di programmazione
  - Errori semantici (più difficili da scoprire!!)
    - Comportamento non aderente alle aspettative/alla intenzionalità del programmatore

L'arte della programmazione

- Validazione
  - Test del programma in tutte le condizioni operative
  - Test su input estremi (es., vettori di dimensione 0 o 1, variabili nulle)
- Documentazione
  - Inserimento di commenti esplicativi nelle varie parti del programma per facilitarne la comprensione (dopo molto tempo dalla stesura o per terze persone)
- Manutenzione
  - Modifica del programma per soddisfare il cambiamento delle specifiche cui deve rispondere

Qualche elemento in più

- Problema
  - Un problema è una situazione che pone delle *domande* alle quali si devono dare delle *risposte*. Risolvere il problema vuol dire uscire dalla situazione
  - Consta dei seguenti elementi:
    - *Dati*: ossia ciò che è noto e che indichiamo col termine *input*
    - *Risultati*: gli elementi incogniti che si devono determinare e che indichiamo con *output*
    - *Condizioni*: le limitazioni cui devono soggiacere i risultati



Qualche elemento in più

- Algoritmo
  - Insieme delle istruzioni che definiscono una sequenza di operazioni mediante le quali si risolve il problema

- Deve essere:
  - finito (numero limitato di passi da compiere in un tempo finito);
  - definito (ogni istruzione deve consentire un'interpretazione univoca);
  - eseguibile (la sua esecuzione deve essere possibile con gli strumenti a disposizione);
  - deterministico (ad ogni passo deve essere definita una operazione successiva)

Un esempio di algoritmo

- Esempio:
  - Problema: Avendo depositato diecimila euro in un conto bancario che produce il 5% di interessi all'anno, capitalizzati annualmente, quanti anni occorrono affinché il saldo del conto arrivi al doppio della cifra iniziale?
  - Algoritmo:
    - 1 L'anno attuale è 0; il saldo attuale è 10.000
    - 2 Ripetere i passi 3 e 4 finché il saldo si mantiene minore di 20.000, poi passare al punto 5
    - 3 Aggiungere 1 al valore dell'anno attuale
    - 4 Il nuovo saldo attuale è il valore del saldo attuale moltiplicato per 1.05 (si aggiunge il 5% al saldo)
    - 5 Il risultato cercato è il valore dell'anno attuale

Formalizzare un algoritmo

- La descrizione delle fasi di un algoritmo può essere formalizzata mediante:
  - formalismi grafici (schemi a blocchi, UML, etc.);
  - pseudocodifica o notazione lineare strutturata: è un linguaggio *semi-formale* con regole, prive di ambiguità ed eccezioni, che esprimono i vari tipi di istruzioni

Esempio: somma di N numeri:

```
inizio
input N
S  = 0
repeat
  input A
  S  = S+A
  N = N-1
until N = 0
output S
fine
```

Pseudolinguaggio…

- …o pseudo codice:
  - È un linguaggio di programmazione fittizio, non è direttamente compilabile ne eseguibile, serve per lo più come "linguaggio di progettazione"
  - Non esiste uno pseduolinguaggio standard, lo scopo è rendere il codice più leggibile → sintassi poco rigorosa
  - E' indipendente dal linguaggio di programmazione
  - Può essere usato per descrivere un algoritmo
  - *Spesso si usano le caratteristiche di C o Pascal come base per definire uno pseudolinguaggio*

Pseudolinguaggio: esempio

1. SE età < maggioreEtà ALLORA

1.1 SE età< etàMINIMA ALLORA

1.1.1 SCRIVI “spiacente non puoi lavorare”

1.2 ALTRIMENTI

1.2.1 SCRIVI “puoi lavorare come apprendista”

1.3 FINE-SE

2. SCRIVI “puoi lavorare”

3. FINE-SE

Dati e istruzioni

- Un programma, in qualunque linguaggio, codifica un algoritmo e consiste dunque di dati e istruzioni
- I dati sono *costanti* o *variabili* durante l'esecuzione del programma
- I dati sono di tipi *pre-definiti* (o primitivi) o *definiti dall'utente* in tutti i linguaggi di oggi
- I dati possono essere distinti in
  - semplici o elementari (interi, reali, caratteri, logici, …)
  - strutturati (vettori, matrici, …) in quanto collezioni di altri dati
- Le istruzioni vengono classificate in
  - dichiarative o non eseguibili: non vengono tradotte e servono per fornire informazioni al traduttore
  - eseguibili: di ingresso/uscita, di controllo, di assegnazione, …

Strutture di controllo

- Nel paradigma della *programmazione strutturata*, le strutture di controllo fondamentali sono tre:
  - sequenza: implicita in tutti i linguaggi di programmazione, l'ordine naturale di esecuzione delle istruzioni è quello sequenziale
  - selezione: *se…allora* oppure *se…allora…altrimenti…*
  - iterazione: *finché…esegui…* oppure *ripeti…finché…*

- Queste tre sole strutture di controllo sono sufficienti a codificare qualunque algoritmo (teorema di Böhm e Jacopini in versione semplificata)

Software

- Il software è l'insieme dei programmi che possono essere eseguiti su una macchina
- Si distinguono tradizionalmente:
  - Software di base
    - Sistema operativo (kernel)
    - Altro software
      - Interprete dei comandi (she
      - Sistemi a finestre (GUI, grap
      - user interface)
      - Software di rete
      - Editor, compilatori, ecc.
  - Software applicativo
    - Programmi che svolgono attività

Software

- Nei sistemi moderni, l'organizzazione del software è *a strati*, ciascuno con funzionalità di livello più alto rispetto a quelli sottostanti
- Nasce così il concetto di macchina virtuale

Sistema operativo

- E' lo strato di programmi che opera al di sopra dell'hardware (e del firmware) e gestisce l'elaboratore consentendone l'uso, nascondendo la complessità e la varietà dell'hardware
- Solitamente, è venduto insieme all'elaboratore
- Spesso si può scegliere tra diversi sistemi operativi per lo stesso elaboratore, con caratteristiche differenti

Sistema operativo

- La parte più interna del SO, detta kernel,
  - gestisce direttamente l'hardware della macchina e le sue periferiche
  - è sempre (quasi tutta) residente in memoria: viene caricata alla accensione del sistema, e ci resta fino al suo spegnimento (o a un crash)
  - fornisce i servizi necessari ai processi in esecuzione attraverso un insieme di *system call*
  - gestisce gli *interrupt* generati dall'hardware

Sistema operativo

- Le principali funzioni messe a disposizione dal S.O. (dipendono anche dalla complessità del sistema di elaborazione):
  - gestione delle risorse (e dei conflitti)
    - gestione della CPU
    - gestione della memoria centrale
    - organizzazione e gestione della memoria di massa
    - gestione della multi-utenza
  - interfaccia utente
    - interpretazione ed esecuzione di comandi.
  - esecuzione dei programmi
  - sicurezza
- Un utente "vede" l'elaboratore solo attraverso il S.O
- E' dunque il S.O. che realizza una "macchina virtuale"

File system

- La parte di sistema operativo che si occupa di gestire i dati su disco è detta *file system*
- Se non ci fosse il file system, l'utente dovrebbe ricordare in quale posizione sul disco è stata inserita ogni singola informazione, sia per accedere a dati già memorizzati, sia per inserirne dei nuovi (posizioni libere)

File system

- Il file system organizza le informazioni in file
- Un file può contenere qualunque tipo di informazione (testi, immagini, moduli oggetto, suoni, filmati, programmi eseguibili, pagine web, …)
- Il file system mantiene una tabella in cui sono indicate:
  - la corrispondenza fra ogni file e le porzioni di disco che il file occupa
  - alcune proprietà dei file (data di creazione/modifica, dimensione, …)
  - le parti di disco disponibili.
- I file sono tipicamente organizzati in directory (o folder, cartelle) gerarchiche