Evoluzione dei linguaggi di alto livello:

Linguaggi di programmazione classificati in base alle loro caratteristiche fondamentali.

Determinano il potere espressivo e si riflettono sullo stile di programmazione.

Linguaggio macchina, binario e fortemente legato all'architettura.

Esecuzione sequenziale, utilizzando una memoria da cui prelevare istruzioni e dati ed inserire risultati.

Linguaggi Assembler, istruzioni del linguaggio macchina e riferimenti alle celle di memoria espressi mediante simboli.

Linguaggi di alto livello, tentativo di astrazione dell'architettura sottostante.

Operazioni di più alto livello rispetto a quelle caratteristiche della macchina.

Evoluzione dei linguaggi di alto livello:

FORTRAN (FORmula TRAnslator) Cobol (Common Oriented Business Language) Basic	esecuzione sequenziale
Pascal Pl/l CLU C	effetto dell'esecuzione: variazione di stato della memoria
Concurrent Pascal Modula CSP Mesa	linguaggi concorrenti
ADA, LISP, INTERLISP ZETA-LISP, MACLISP ML, SASL	esecuzione come calcolo del valore di una funzione
PROLOG M-PROLOG PROLOG2 QUINTUS PROLOG	esecuzione come dimostrazione della verità di una formula logica
CONCURRENT PROLOG PARLOG FLAT CONCURRENT PROLOG	
SMALLTALK C++ EIFFEL FLAVORS	data encapsulation esecuzione attraverso scambio di messaggi

Fattori evolutivi dei linguaggi:

Lo sviluppo dei linguaggi di programmazione ha seguito un lungo, continuo, processo governato da vari fattori. Alcuni dei più imporanti:

- **Hardware**
  Influenza delle macchine fisiche sui linguaggi.
- **Applicazioni**
  Le applicazioni dei calcolatori, inizialmente di tipo solo numerico, si sono rapidamente estese a capi diversi, anche non numerici. Nuovi campi d'applicazione richiedono linguaggi con caratteristiche specifiche. Ad es. linguaggi per la manipolazione di simboli nell'ambito dell'IA e della gestione della conoscenza.
- **Nuove metodologie**
  Sviluppo di nuove metodologie di programmazione, come ad es. il paradigma della programmazione orientata agli oggetti.
- **Teoria**
  Ruolo importante degli studi teorici per selezionare alcune tipologie di costrutti e per migliorare l'attività di programmazione. Es. eliminazione del *goto*.

Anni '50-60: FORTRAN, ALGOL, LISP, COBOL, SIMULA
Anni '70: C, PASCAL, SMALLTALK, PROLOG
Anni '80: C++, ADA
Anni '90: JAVA

Una possibile classificazione:

- IMPERATIVI
- FUNZIONALI
- LOGICI
- AD OGGETTI

A loro volta possono essere SEQUENZIALI o CONCORRENTI.

Perché tanta proliferazione di linguaggi?

Perché tante sono le differenti **applicazioni**.

- Efficienza;
- Rapida prototipazione;
- Facile comprensione del linguaggio (leggibilità);
- Modificabilità e Riusabilità;
- Programmazione in largo;
- Qualità del programma;
- ecc.

Linguaggi imperativi

Le caratteristiche essenziali sono dettate dalla architettura di Von Neumann (processore e memoria).

Tutti i linguaggi tradizionali (ASSEMBLER, FORTRAN, Pascal) sono livelli di astrazione costruiti sopra tale architettura.

L'architettura consiste di due parti:
memoria (passiva)
processore (attivo)

La principale attività del processore è assegnare valori a celle di memoria.

Variabili come astrazione di celle di memoria ed istruzione di assegnamento.

I linguaggi imperativi adottano uno **stile prescrittivo.**

Il programma imperativo prescrive le operazioni che il processore deve compiere (es. assegnamento);

Esecuzione delle istruzioni nell'ordine in cui appaiono nel programma (eccezione: strutture di controllo).

Realizzati sia attraverso interpretazione (BASIC,...) che compilazione (Pascal, FORTRAN, COBOL,...)

Sono forse la classe di linguaggi più matura;

Nati per manipolazione numerica più che simbolica;

Sono evoluti verso:

- blocchi (ALGOL);
- moduli (MODULA);
- tipi di dato astratto (SIMULA)
- parallelismo (Concurrent Pascal, CSP, etc.)

in generale verso costrutti adatti per programmazione in largo (programmazione ad oggetti).

Sono generalmente linguaggi che adottano un controllo statico (stretto) di tipo.

Sono generalmente efficienti.

Esempio (Pascal)

```
program mult(input, output);
var first, second, molt: integer;
begin
   read(first,second);
   molt=first*second;
   write(molt)
end.
```

programma = algoritmo + dati

La struttura del programma consiste in un'intestazione in cui si assegna il nome al programma seguita da due parti:

1. Una parte di dichiarazione in cui si dichiarano tutte le variabili del programma ed il loro tipo;

2. Una parte istruzioni che descrive l'algoritmo risolutivo utilizzato, mediante istruzioni del linguaggio.

Le istruzioni si dividono in:

1. Istruzioni di **lettura** e **scrittura**;
2. Istruzione di **assegnamento** (astrazione della cella di memoria);
3. Istruzioni di **controllo**.

Programmazione modulare[1]

Inizialmente intesa come costruzione di programmi assemblando parti, di solito sotto-programmi.

Tecnica di suddividere un progetto software in parli il più possibile indipendenti (moduli sviluppabili ***separatamente***, con compilazione e testing separati), le cui modalità di interazione siano ben definite (***interfacce*** standard).

Non e' necessario conoscere i dettagli dell'implementazione.

Supporto a progettazione sia ***top-down*** che ***bottom-up***.

Permette di ottenere sia riusabilità che estendibilità .

==> concetto di modulo

Presente in linguaggi quali Modula (Wirth), Pascal (TurboPascal), MESA, Ada.

[1] A. NATALI, "Introduzione alla costruzione del software", Esculapio, 1993.

Prime estensioni in linguaggi tradizionali:

Sviluppo di codice separato:

Meccanismi che operano a ***livello di codice sorgente***, ed intervengono ***prima della compilazione***.

- possibile incapsulamento di insiemi di istruzioni (MACRO) espanse quando servono;
- riutilizzo di parti di codice espresse in sorgenti già disponibili includendoli in nuovi programmi

Ad esempio: nel linguaggio C e' possibile includere file sorgenti.

In TurboPascal, esiste una direttiva al compilatore che consente di ***includere il codice sorgente*** contenuto in un altro file nel programma:

program main (input, output);
{$I utivetr}
var V:vettore; ...

Il codice sorgente contenuto nel file **utivetr.pas** viene incluso nel punto in cui si trova la direttiva.

Si possono avere più inclusioni di file sorgenti annidate (fino a 15 nel TurboPascal ver. 6)

Meccanismi che operano a ***livello di codice oggetto***, ed intervengono ***durante la fase di collegamento*** (linking)

- dichiarazioni di importazione ed esportazione di entità fra file diversi collegati a formare un unico programma.

In TurboPascal la creazione di librerie collegate (***link***) ai vari programmi (senza renderne disponibile il codice sorgente) si ottiene attraverso il costrutto ***unit***.

Esistono ***unit*** predefinite.

La filosofia dell'uso di ***unit*** può essere seguita in modo sistematico anche nello sviluppo di programmi di utente.

====> ***programmazione modulare***

Programmazione modulare: requisiti

Occorre che siano soddisfatti alcuni principi:

1. Moduli corrispondenti ad unità sintattiche nel linguaggio usato (separatamente compilabili);

2. Ciascun modulo deve "comunicare" con il minor numero di moduli (solo con quelli necessari);

3. Se due moduli si interfacciano tra loro, le loro interfacce devono scambiarsi il minor numero di informazioni (solo con quelle necessarie);

4. Se due moduli comunicano tra loro, questo deve essere evidente dal loro codice;

5. Le informazioni di un modulo sono private, a meno che il modulo non le dichiari esplicitamente pubbliche (***information hiding***).

Concetto di modulo:

Un ***modulo*** raggruppa al suo interno un insieme di informazioni (tipi, costanti, variabili, procedure e funzioni)

In generale conterrà ***dati*** ed ***operazioni***.

Solo ciò che è [esplicitamente] esportato all'esterno del modulo e dualmente solo ciò che è [esplicitamente] importato è disponibile all'interno di un modulo

Un modulo perciò definisce e confina un preciso ***ambiente di visibilità***.

I riferimenti fra moduli sono risolti staticamente durante la fase di collegamento ==> il programma eseguibile non distingue più le unità separate

Esempio (TurboPascal):

Le entità ***esportate*** sono definite in una parte dichiarativa dell'unità detta interfaccia.

```
interface
   const  c1 = 'messaggio di prova';
   type   t1 = array [1..30] of string;
   var    v1 : t1;
   procedure p (X,Y: integer); {solo intestazione}
   function…;
```

Le entità ***importate*** sono determinate mediante la parola chiave ***uses***.

```
program Esempio1;
uses M1;
begin
  write(c1);
  p(A,B);
  …
end.
```

```
unit M1;

interface
    const   c1 = 'messaggio di prova';
    type    t1 = array [1..30] of string;
    var     v1 : t1;
    procedure p (X,Y: int.eger); {solo intestazione}

implementation
    procedure p (X,Y: integer);
    {codice della procedura}

begin
{inizializzazione delle variabili di Ml}

V1:='     '
end.
```

Sintassi delle unit TurboPascal:

```
unit <nome>
    interface
    [uses <lista-unit>]
    [<dichiarazioni esportate>]
    implementation
    [uses <lista-unit>]
    [<dichiarazioni locali (nascoste»]
    [begin
        <istruzioni di inizializzazione> ]
    end.
```

Le sezioni *interface* ed *implementation* di una unit riflettono l'idea di modulo come componente software dotato di interfaccia nota ed accessibile ed implementazione nascosta.

Componenti software mediante moduli:

Il modulo (unit) è una unità sintattica con regole di visibilità dei nomi.

Si presta quindi a racchiudere le definizioni e le operazioni che realizzano un’astrazione di dato.

Ci sono diversi livelli di utilizzo di moduli per realizzare:

Librerie:
Il modulo rende visibili procedure e funzioni che non fanno uso di variabili non locali. Il modulo è una collezione di **operazioni** (ad esempio, funzioni matematiche).

Astrazioni di dato:
Il modulo ha dati locali (nascosti) e rende visibili all'esterno le operazioni invocabili (procedure e funzioni) su questi dati locali, ma non gli identificatori dei dati. La parte di inizializzazione del modulo può assegnare un valore iniziale ai dati locali nascosti.

Tipo di dato astratto:
Il modulo esporta un identificatore di tipo T e le operazioni eseguibili su dati dichiarati di questo tipo. I "clienti" del modulo dichiarano e controllano quindi il tempo di vita delle variabili di tipo T.

Linguaggi imperativi: conclusioni

Sono i linguaggi che consentono un migliore utilizzo della macchina di von Neumann (efficienza) ma ...

Scarsa naturalezza e leggibilità;

Scarsa flessibilità;

Programmi, procedure, funzioni non sono manipolabili da programma;

Non sono adatti per la manipolazione di simboli e sequenze di simboli (es. liste), ma piuttosto per elaborazione numerica;

Le funzioni non sono funzioni matematiche **(effetti collaterali);**

Le variabili non sono variabili matematiche;

Difficili prove formali di correttezza e la verifica di programmi;

Non facilitano la programmazione incrementale.

Scelta di nuovi modelli

Fattori predominanti:

- Applicazioni di Intel1igenza Artificiale;
- Programmazione esplorativa e prototipale;
- Crescente crisi del software: tentativo di sviluppare programmi più concisi, più semplici da scrivere, più vicini alla logica del problema, più semplici da verificare formalmente;
- Sviluppo di VLSI e nuove architetture: grandi collezioni di processori operanti in parallelo;

Perchè una architettura deve dettare le specifiche ad un linguaggio di programmazione?

Le specifiche di un linguaggio ad alto livello possono determinare una nuova architettura (LISP machine).

==> programmazione funzionale

==> programmazione logica.

Caratteristiche comuni:

Non manipolazione di numeri, ma di simboli;

Il "programma" è costantemente in evoluzione:

- modularità spinta
- programma come struttura dati;
- 1'ambiente tende a confondersi col linguaggio.

La programmazione e' esplorativa ed incrementale: prototipi veloci;

Il linguaggio deve essere ad "altissimo livello": utilizzabile anche da non-programmatori.

Applicazioni di Intelligenza Artificiale (riconoscimento di immagini, apprendimento automatico, etc.).

Linguaggi funzionali (o applicativi)

Concetto primitivo: quello di **funzione** e di **applicazione di funzione**.
La caratteristica fondamentale dei linguaggi di questo paradigma, almeno nella versione *pura*, è quella di non possedere il concetto di memoria (e dunque di effetto collaterale).
Una **funzione** e' una regola di corrispondenza che associa ad *ogni* elemento del suo dominio un **unico** elemento nel codominio.

Una definizione di funzione specifica il **dominio**, il **codominio** e la **regola di associazione**.

incr(x)::=x + 1.

Dopo la definizione una funzione puo' essere applicata ad un elemento del dominio per restituire 1'elemento associato del range (valutazione):

incr(3) -> 4.

Linguaggi funzionali (segue)

- La sola OPERAZIONE del modello funzionale e' l'APPLICAZIONE di FUNZIONI ad OPERANDI;
- Il ruolo della macchina FUNZIONALE (interprete) e' VALUTARE il programma e produrre un valore;
- Il valore di una funzione e' determinato solo dai suoi argomenti (assenza di effetti collaterali, funzionale puro);
- L'essenza della programmazione funzionale e' COMBINARE funzioni (utilizzo della ricorsione);
- Le VARIABILI sono variabili MATEMATICHE che denotano un valore fisso nel tempo e non valori mutabili (nessun assegnamento)

programma = funzione

Esecuzione del programma = valutazione della funzione

Lavorano su alberi binari: liste

Anche i programmi sono strutture dati

Il linguaggio LISP (LISt Processing)

Nato nel 1958 ad opera di J. McCarthy;

Originariamente era puramente funzionale;.

LISP oggi non vuole dire solo linguaggio, ma anche ambiente di programmazione (LISPMACHINE)

LISP ha un ricchissimo ambiente di programmazione ed un'ampia comunita' che lo utilizza:

InterLisp
MacLisp
CommonLisp
…

```
(defun member   (item list)
            (cond. ((nulilist) nil)
                        ((equal item (car list)) T)
                        (T (member item Ccdr list)))))
```

```
(member 'A '(B C D))
```

Il programma e' una struttura dati;

Non c'e' dichiarazione di tipo;

Non c'e' assegnamento.

Quando usare i linguaggi funzionali:

Manipolazione simbolica e non numerica;

Applicazioni di Intelligenza Artificiale;

Programmazione esplorativa;

Modifica dinamica dei programmi;

Problemi naturalmente ricorsivi;

Non per problemi di controllo di processo o real-time.

E' un linguaggio da cui si può imparare molto sulla costruzione di algoritmi ed, in generale, sulla realizzazione dei linguaggi (legami di variabili; scope rules).

Un interprete di LISP puro puo' essere scritto in LISP molto semplicemente (EVAL).

La macchina di von Neumann si perde di vista anche se LISP -> costrutti per assegnamento e iterazione.

Linguaggi logici: programmazione dichiarativa

programma = conoscenza + controllo

La conoscenza sul problema e' espressa indipendentemente dal suo utilizzo (COSA non COME);

Alta modularità e flessibilità;

E' lo schema progettuale di gran parte dei sistemi basati sulla conoscenza (Sistemi Esperti);

I moderni linguaggi di programmazione per Intelligenza Artificiale tendono a riprodurre tale schema;

Definire un linguaggio in tale schema significa definire come il programmatore puo' esprimere la conoscenza e quale tipo di controllo puo' utilizzare;

Problematiche di RAPPRESENTAZIONE della conoscenza;

Linguaggi per la rappresentazione della conoscenza:

KEE, OPS-5, MRS, FRL .....

DICHIARATIVO - PROCEDURALE

Esempio: Ordinamento di una lista

DICHIARATIVO:

"Il risultato dell'ordinamento di una 1ista vuota e' la lista vuota"

"Il risultato dell'ordinamento di una lista L è L' se la lista L' è ordinata ed è la permutazione di L"

PROCEDURALE:

"Controlla prima se la lista e' vuota; se si dai come risultato la lista vuota"

"Altrimenti calcola una permutazione L' di L e controlla se e' ordinata; se si, termina dando come risultato L', altrimenti calcola un'altra permutazione di L etc…"

Il linguaggio PROLOG (PROgramming in LOGic)

Nasce nel 1973;

Si fonda sulle idee di programmazione logica avanzate da R. Kowalski;

E' rimasto confinato a pochi circoli accademici;

E' il piu' noto linguaggio di programmazione dichiarativo:

algoritmo = logica + controllo

Algoritmi equivalenti:

A1 = L + Cl
A2 = L + C2

Si basa sulla logica dei predicati del prim'ordine (prova automatica di teoremi - Risoluzione);

Le strutture dati su cui lavora sono alberi e anche i programmi sono strutture dati manipolabili;

Utilizzo della ricorsione, assenza di assegnamento.

Un esempio (PROLOG):

member(Item,[Item, Rest]).

member(Item,[First, Rest]) :- member(Item,Rest).

? - member(a,[b,a,c]).
? - member(Item, [b,a,c]).
?- member(a,List).

In esso sono potenzialmente nascosti tanti programmi diversi.

NON-RIDONDANZA

INVERTIBILITA'

VARIABILI A SINGOLO ASSEGNAMENTO

Programma PROLOG

Un programma PROLOG e' un insieme di clausole di Horn (formule logiche), che rappresentano:

- fatti, riguardanti gli oggetti del dominio e le loro relazioni;
- regole sugli oggetti e sulle relazioni;
- goal o interrogazione sulla base di conoscenza (programma) definita.

Un goal viene dimostrato provando i singoli sotto-goal (atomi) da sinistra a destra.

Il sotto-goal selezionato viene provato confrontandolo con le teste delle clausole contenute nella base di conoscenza.

Un esempio (PROLOG)

"Due individui .sono colleghi se lavorano nella stessa ditta"

```
collega(X,Y):-
        lavora(X,Z),lavora(Y,Z), X<> Y.

lavora(lamma,deis).
lavora(mello, deis).
lavora(corradi, deis).
lavora(gorrieri, dm).
lavora(asperti,dm).

? -collega(lamma,mello).
yes

? -collega(X, Y).
X=lamma Y =mello;
X=lamma Y=corradi;
X=gorrieri Y =asperti;
X=mello Y =lamma.
…
```

Quando utilizzare un linguaggio logico:

Manipolazione simbolica e non numerica;

Applicazioni di A.I.;

Modifica dinamica dei programmi;

Problemi naturalmente ricorsivi.
(condividono caratteristiche coi linguaggi funzionali)

Metodologicamente ci si concentra piu' sulla specifica del problema che non sulla strategia di soluzione (efficienza).

BASE LOGICA

LOGICA: base di molti concetti e discipline.

PROBLEMI:

Sono linguaggi relativamente giovani:

MANCANZA DI AMBIENTI DI PROGRAMMAZIONE EVOLUTI .

EFFICIENZA: Paragonabile a quella del LISP

Per programmi complessi con particolari vincoli la programmazione dichiarativa e' ancora un'utopia.

Programmazione ad oggetti:

IN ESSI LINGUAGGIO ED AMBIENTE DI PROGRAMMAZIONE SI CONFONDONO

Icone ed interfaccia a finestre.

In alcuni moderni ambienti di programmazione sono presenti concetti analoghi a quelli del modello ad oggetti.

Tali ambienti di programmazione sono in generale orientati allo sviluppo incrementale delle applicazioni.

Il più noto e' il linguaggio C++

Origine del modello ad oggetti:

All'origine del concetto di oggetto c'e' la necessita' di rispondere a due esigenze distinte: quella di MODULARITA' e quella di ASTRAZIONE.

Linguaggi di Programmazione

Sistemi Operativi

Intelligenza Artificiale

Esigenza di una metodologia di strutturazione dei programmi basata sul riutilizzo delle informazioni

EVOLUZIONE DEI LINGUAGGI DI PROGRAMMAZIONE (IMPERATIVI)

Introduzione di costrutti che permettono la decomposizione in MODULI di informazione

==> CONCETTO DI MODULO

Introduzione di costrutti di astrazione con sufficiente capacita' espressiva

==> CONCETTO DI TIPO DI DATO ASTRATTO

Entita' descrittiva ("template") che specifica le caratteristiche (rappresentazione dei dati ed operazioni) di ogni astrazione di dato generata da esso.

==> CONCETTO DI OGGETTO

CONCETTI BASE:

Oggetto e Messaggio

Cambiamento del punto di vista:

Calcolo orientato ai dati.

Ereditarietà (gerarchie di oggetti)

Oggetti = Classi o Istanze

Proprietà del modello ad oggetti:

ASTRAZIONE:

OGGETTO come contenitore informazioni + interfaccia;

STATO + OPERAZIONI
INTERFACCIA (metodi invocabili dall'esterno)

CLASSIFICAZIONE:

Relazioni classe/istanza (meta-descrizioni);

EREDITARIETA':

Relazioni di ereditarietà tra le classi, per ottenere riutilizzo di comportamenti comuni;

DINAMICITA':

Creazione dinamica di nuovi oggetti o variazioni del comportamento.

Programmazione ad oggetti

- Linguaggi ad oggetti si sono evoluti a partire dal concetto di astrazione di dato, tipo di dato astratto e classe introdotti in linguaggi imperativi;
- Punto di unione fra programmazione tradizionale e rappresentazione della conoscenza;
- Adatti per la programmazione "in largo";
- RIUSABILITA';
- Incrementalità: ciascuna classe estende le definizioni delle sue superclassi;
- Interfaccia ed ambiente gradevole.