Corso di Linguaggi di Programmazione

Lezione 1

Alberto Ceselli
alberto.ceselli@unimi.it

Università degli Studi di Milano

27 Febbraio 2013

- Docente:
  Alberto Ceselli alberto.ceselli@unimi.it
- Orario del corso:
  - mercoledì dalle 14.00 alle 16.00
  - venerdì dalle 11.00 alle 13.00
- Ricevimento: TBA (mercoledì dopo la lezione, ma anche in altri momenti, previo appuntamento)
- Homepage:
  www.dti.unimi.it/linguaggi
  www.dti.unimi.it/∼ ceselli/linguaggi

- Libro di riferimento: Sebesta R.W. Concepts of programming languages, Pearson / Addison Wesley
- (è disponibile l'*ottava* edizione)
- Riferimenti più specifici forniti durante il corso

- Modalità di esame:
  - esercizi durante il corso
  - progetto + relazione
  - prova orale

- Libro di riferimento: Sebesta R.W. Concepts of programming languages, Pearson / Addison Wesley
- (è disponibile l'*ottava* edizione)
- Riferimenti più specifici forniti durante il corso

- Modalità di esame:
  - esercizi durante il corso
  - progetto + relazione
  - prova orale

- Altre domande sull'organizzazione?

La nascita dei computer e dei Linguaggi di Programmazione (LdP)

- Chi ha progettato il primo computer?

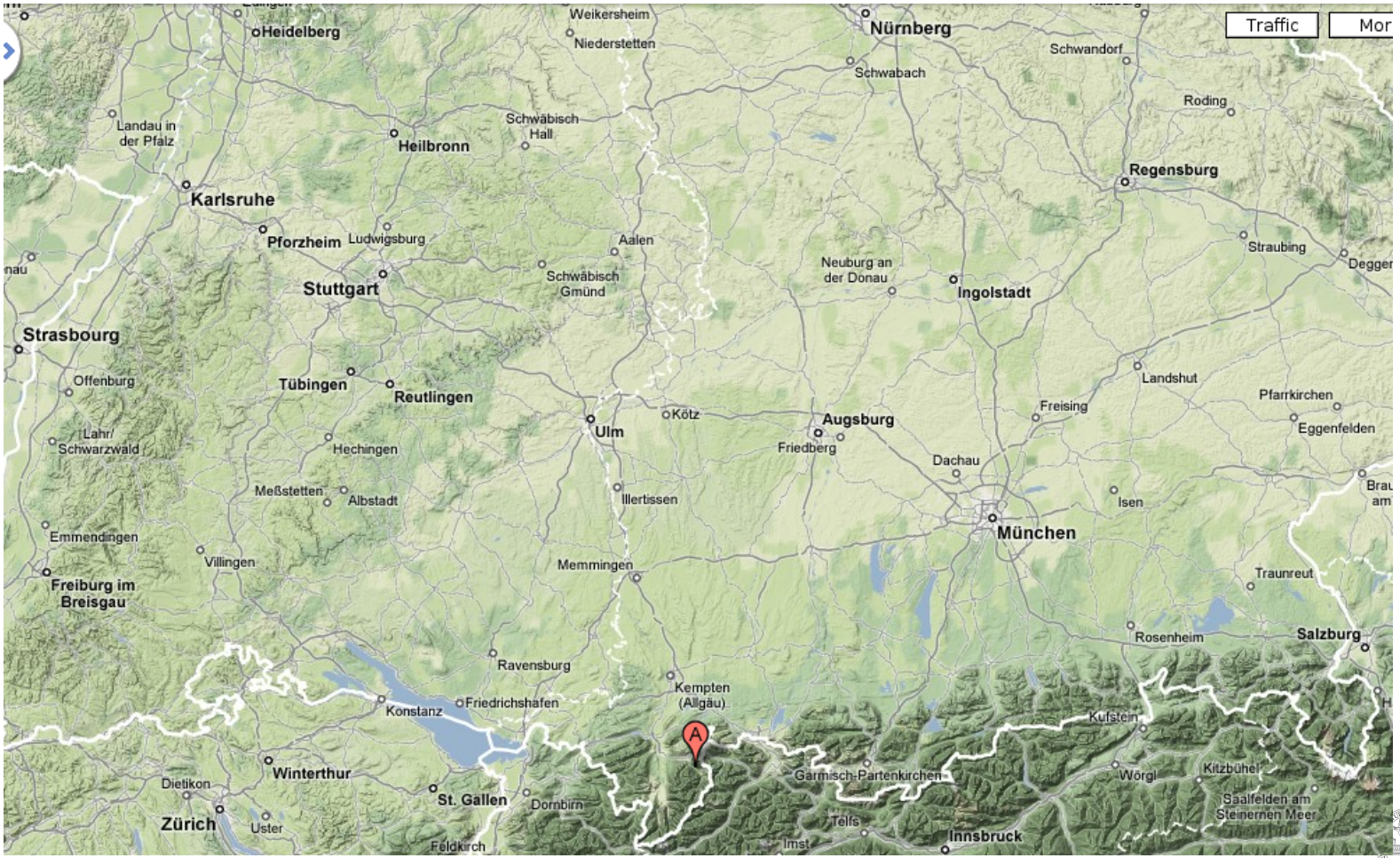
La nascita dei computer e dei Linguaggi di Programmazione (LdP)

- Chi ha progettato il primo computer?
- Konrad Zuse!

La nascita dei computer e dei Linguaggi di Programmazione (LdP)

- Chi ha progettato il primo computer?
- Konrad Zuse!
- ha progettato il primo linguaggio di programmazione (tesi di PhD)
- Berlino, durante la II guerra mondiale
- si è spostato (da solo) a Hinterstein nel '45 con l'unico prototipo non distrutto (Z4)

HinterStein



HinterStein

Konrad Zuse e Plankalkül

Porzione di programma in Plankalkül:

	A + 1	=>	A
V	4		5
S	1.n		1.n

che equivale a $A[5] := A[4] + 1$, con $A[4]$ ed $A[5]$ interi di n bit.

- Prima riga: istruzione
- Seconda riga (V): subscripts (opzionale)
- Terza riga (S): tipi

Konrad Zuse e Plankalkül

Piuttosto completo, con caratteristiche "avanzate" rispetto ai suoi successori

- Tipi di dati: bit, interi, float con "hidden bit"

Konrad Zuse e Plankalkül

Piuttosto completo, con caratteristiche "avanzate" rispetto ai suoi successori

- Tipi di dati: bit, interi, float con "hidden bit"
- Array e record (incluse definizioni ricorsive)

Konrad Zuse e Plankalkül

Piuttosto completo, con caratteristiche "avanzate" rispetto ai suoi successori

- Tipi di dati: bit, interi, float con "hidden bit"
- Array e record (incluse definizioni ricorsive)
- Niente goto, cicli for (simile ad ADA), statement Fin con apice (continue / break per cicli annidati)

Konrad Zuse e Plankalkül

Piuttosto completo, con caratteristiche "avanzate" rispetto ai suoi successori

- Tipi di dati: bit, interi, float con "hidden bit"
- Array e record (incluse definizioni ricorsive)
- Niente goto, cicli for (simile ad ADA), statement Fin con apice (continue / break per cicli annidati)
- Costrutto di selezione (if), ma senza else

Konrad Zuse e Plankalkül

Piuttosto completo, con caratteristiche "avanzate" rispetto ai suoi successori

- Tipi di dati: bit, interi, float con "hidden bit"
- Array e record (incluse definizioni ricorsive)
- Niente goto, cicli for (simile ad ADA), statement Fin con apice (continue / break per cicli annidati)
- Costrutto di selezione (if), ma senza else
- Espressioni matematiche per relazioni tra variabili (gestione invarianti, tipo assert)

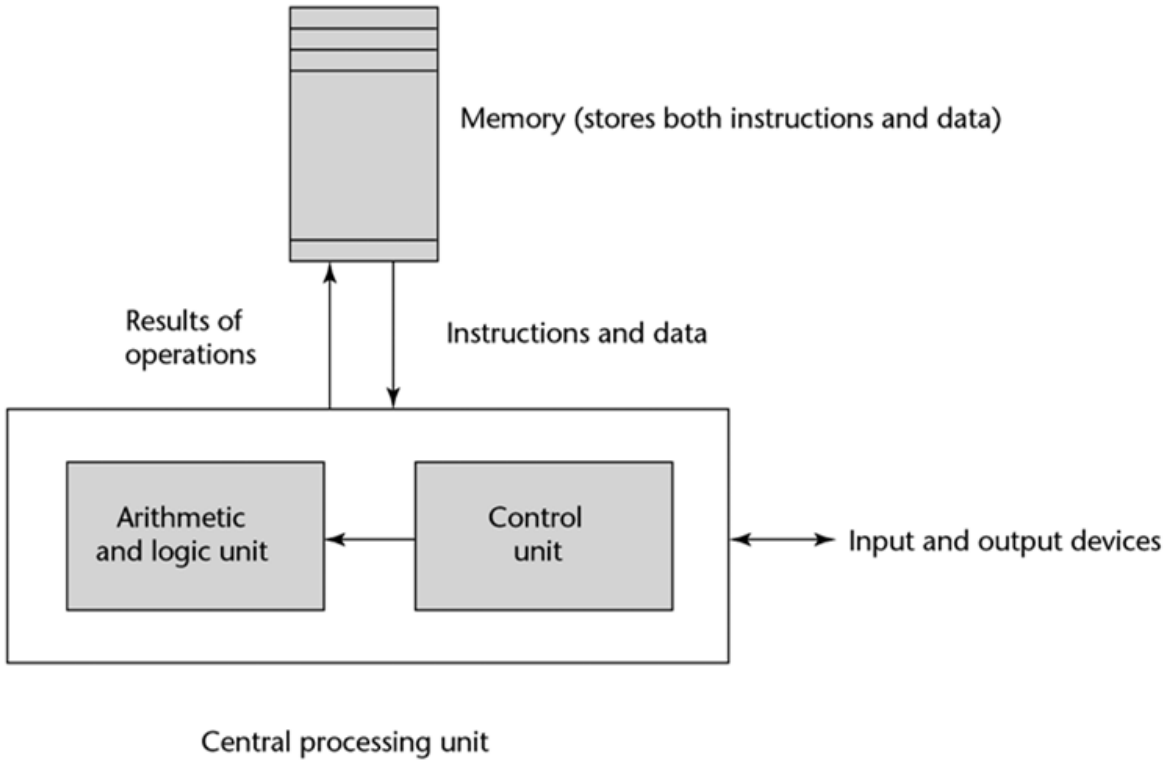
Konrad Zuse e Plankalkül

- Ordinamento di un array
- Connettività di grafi
- Calcolo floating point (es. radici quadrate)
- Analisi sintattica di formule logiche
- 49 pagine di algoritmi per gli scacchi (anche se non era un esperto)

Von Neumann, linguaggio macchina e short code

Figure 1.1

The von Neumann computer architecture



Von Neumann, *linguaggio macchina* e short code

Porzione di programma in linguaggio macchina:

```
00000010101111001010
00000010111111001000
00000011001110101000
```

Somma dei valori memorizzati nelle locazioni di memoria 10 e 11, e memorizzazione alla locazione 12.

Von Neumann, linguaggio macchina e *short code*

Short code:

01	?	06	abs value	1n	potenza (n + 2)-esima
02	)	07	+	2n	radice (n+2)-esima
03	=	08	pause	4n	if $<= n$
04	/	09	(	58	print e tab

- X0 = SQRT (ABS (Y0))
- codificato come '00 X0 03 20 06 Y0'
- Puramente interpretato (o tradotto *a mano*).
- Anche altri linguaggi simili (assembly) sono stati "implementati" e "tradotti" in linguaggio macchina.

... vs Python

Porzione di programma Python:

```
def intersect (seq1, seq2):
    res = []
    for x in seq1:
        if x in seq2:
            res.append(x)
    return res

...

>>> s1 = "SPAM"
>>> s2 = "SCAM"
>>> intersect(s1, s2)
['S','A','M']
```

... vs FRACTAN (Conway)

Porzione di programma Fractan: (thanks to M. Fiorentini)
Crivello di Eratostene (ricerca di numeri primi):

$$\frac{17}{91}, \frac{78}{85}, \frac{19}{51}, \frac{23}{38}, \frac{29}{33}, \frac{77}{29}, \frac{95}{23}, \frac{77}{19}, \frac{1}{17}, \frac{11}{13}, \frac{13}{11}, \frac{15}{14}, \frac{15}{2}, \frac{55}{1}$$

Interprete: inizia con 2, e ad ogni passo moltiplica l'ultimo numero ottenuto per la prima frazione che renda intero il risultato; quando ottieni una potenza di 2, prendi il suo esponente: è primo. versione ottimizzata:

$$\frac{3}{11}, \frac{847}{45}, \frac{143}{6}, \frac{7}{3}, \frac{10}{91}, \frac{3}{7}, \frac{36}{325}, \frac{1}{2}, \frac{36}{5}$$

Interprete: inizia con 10 e considera le potenze di 10.

... vs Piet

Porzione di programma Piet: (thanks to R. Cordone)

hello world:

the towers of Hanoi:

http://www.dangermouse.net/esoteric

Il ruolo dei linguaggi di programmazione

- "the art of programming is the art of organising complexity"
- "we must organise the computations in such a way that our limited powers are sufficient to guarantee that the computation will establish the desired effect."
- "program testing can be used to show the presence of bugs, but never to show their absence!"

(Dijkstra)
In sintesi: non c'è speranza di stabilire la correttezza di un programma tramite testing ...
... a meno che non si tenga conto anche della sua struttura interna.

Programming domains

- Calcolo scientifico
- Business
- Intelligenza artificiale
- Programmazione di sistemi operativi
- Scripting e sviluppo web
- ...

Motivazione

- Migliorare la capacità di esprimere idee
- Migliorare il background per scegliere i linguaggi più opportuni
- Migliorare la capacità di imparare nuovi linguaggi
- Comprendere meglio il significato dell'implementazione
- Visione generale dello sviluppo del calcolo

Programma del corso

- T Potenzialità, sintassi e semantica di un linguaggio di programmazione
- V paradigma funzionale
- V paradigma dichiarativo
- T costrutti per la strutturazione dei dati
- T costrutti per la strutturazione della computazione
- T costrutti per la strutturazione dei programmi
- T object orientation e generic programming
- T costrutti per la gestione della concorrenza

Programma del corso

- Panoramica storica dell'evoluzione dei linguaggi di programmazione; caratteristiche comuni e classificazione dei linguaggi di programmazione; costi legati alla scelta di un linguaggio di programmazione.
- Il problema della verifica automatica della correttezza di programmi; richiami ai modelli di calcolo astratti; indecidibilità algoritmica.
- Descrizione formale della sintassi e della semantica di un linguaggio di programmazione. Generatori automatici di analizzatori lessicali e sintattici.

- Paradigma funzionale: ricorsione vs iterazione, ricorsione in coda; programmazione con un linguaggio funzionale.
- Paradigma dichiarativo: la programmazione logica e l'interprete PROLOG; programmazione logica a vincoli.

Programma del corso

- Tipi: Definizione, rappresentazione interna, operazioni, compatibilit. Tipi elementari, tipi strutturati e user defined. Sottotipi, tipi astratti, generici e parametrici. Polimorfismo.
- Gestione della memoria.
- Meccanismi per la strutturazione dei programmi e supporto all'ingegneria del software; information hiding, object orientation, encapsulation, inheritance e polimorfismo; programmazione generica; librerie, riutilizzo e portabilit del software.
- Costrutti per la programmazione concorrente; la concorrenza nei linguaggi imperativi.
- Eccezioni: definizione, generazione e gestione. Programmazione ad eventi.

Criteri

Alcuni *criteri* per la classificazione di un linguaggio di programmazione:

- Leggibilità (macchina o uomo?)
- Facilità di sviluppo
- Affidabilità

Criteri

Alcuni *criteri* per la classificazione di un linguaggio di programmazione:

- Leggibilità (macchina o uomo?)
- Facilità di sviluppo
- Affidabilità
- *Costo*

Caratteristiche

Alcune *caratteristiche* per la classificazione di un linguaggio di programmazione:

- Semplicità, ortogonalità
- Strutture dati e controllo del flusso
- Design e sintassi
- Factoring e locality
- Supporto per l'astrazione, flessibilità
- Type checking, gestione eccezioni e controllo aliasing

semplicità

- numero di componenti del linguaggio: se il linguaggio ha molte componenti, ogni programmatore acquisice ed usa solo un sottoinsieme del linguaggio; questi sottoinsiemi, in generale, variano da programmatore a programmatore: è difficile leggere codice scritto da altri
- molteplicità delle features (es. incrementi in C o Java)
- pochi, semplici costrutti aiutano a sfruttare tutte le potenzialità del linguaggio
- overloading degli *operatori* (es. operatori '+' e '*' tra arrays)
- assembly è perfetto!

ortogonalità

- def. **ortogonalità:** un insieme (relativamente) piccolo di costrutti primitivi possono essere combinati in un numero (relativamente) piccolo di modi per costruire il controllo e le strutture dati del linguaggio
- inoltre, ogni possibile combinazione di primitive è legale e sensata
- il significato di una feature di un linguaggio ortogonale *è indipendente dal contesto* in cui appare nel programma
- più un linguaggio è ortogonale, meno eccezioni si presentano.
- se ogni combinazione è legale, si possono effettuare meno controlli sintattici
- esempio di linguaggio NON ortogonale: il C
- esempio di linguaggio ortogonale: LISP

controllo del flusso

- while, for, repeat ... until migliorano la leggibilità del programma rispetto a goto
- in generale, è più semplice leggere un programma dall'inizio alla fine che seguire l'ordine di esecuzione di goto

strutture dati

- esempio 1: tipi Boolean
- esempio 2: tipi record (vettori di record o insiemi di vettori)

sintassi

- restrizioni sulla forma degli identificatori
- forma delle parole chiave (es. begin-end, { e } oppure if-endif); parole chiave che possono anche essere utilizzate come identificatori
- legame tra forma e significato
(es. static in C, g/<regular-expression>/p in ed)

Scope

- **factoring**: flessibilità offerta per suddividere un programma in unità separate (es. sottoprogrammi, definizioni simboliche di costanti, macro ...)
- **locality**: l'effetto di un'istruzione deve essere ristretto ad una porzione breve e circostante del codice di un programma

supporto per l'astrazione

- brevemente **astrazione**: abilità di definire e poi utilizzare strutture od operazioni complicate, in modo che molti dettagli possano essere ignorati
- es. utilizzo di sottoprogrammi (process abstraction)
- es. utilizzo di record per rappresentare nodi di un albero (data abstraction)

espressività

- operatori potenti permettono di implementare funzioni complesse con programmi compatti.
- es. incremento in C
- es. for invece che while

Affidabilità

- factoring e locality (posso agire selettivamente su moduli)
- type checking (specialmente a compile time)
- exception handling (abilità di un programma di intercettare errori a run time)
- aliasing (due o più reference alla stessa locazione di memoria, es. union o puntatori in C)
- facilità di lettura e scrittura del codice

Costo per la programmazione

- addestramento dei programmatori:
  - semplicità e dell'ortogonalità del linguaggio
  - esperienza dei programmatori
  - linguaggi più potenti sono spesso più difficili da imparare (anche se non dovrebbe essere!)
- costo per l'implementazione:
  - funzione della facilità di sviluppo
  - il linguaggio è progettato in modo particolare per l'applicazione?
- disponibilità di ambienti di sviluppo evoluti

Costo per la compilazione e l'esecuzione

- realizzazione o disponibilità di compilatori / interpreti a costo contenuto
- compilatori ottimizzanti / 'just in time'
- esecuzione del compilatore (necessità di macchine particolari)
- esecuzione del programma

Costo per la manutenzione del software

- affidabilità
  - sistemi critici (aerei, impianti nucleari ...)
  - supporto clienti
- aggiornamento (2X – 4X rispetto allo sviluppo)

Altro

Ci sono molte altre caratteristiche di un LdP di cui tener conto (a seconda del tipo di applicazione):

- portabilità – mobilità
- generalità

In ordine, i criteri più importanti nella valutazione di un LdP sono:

1. facilità di sviluppo
2. leggibilità
3. affidabilità

Criteri di valutazione e caratteristiche di un LdP

	Criteri		
Caratteristica	**Leggibilità**	**Sviluppo**	**Affidabilità**
semplicità – ortogonalità			
strutture di controllo			
tipi e strutture dati			
design e sintassi			
factoring			
locality			
supporto all'astrazione			
espressività			
type checking			
gestione delle eccezioni			
restrizione dell'aliasing			

Criteri di valutazione e caratteristiche di un LdP

	Criteri		
Caratteristica	**Leggibilità**	**Sviluppo**	**Affidabilità**
semplicità – ortogonalità	X	X	X
strutture di controllo	X	X	X
tipi e strutture dati	X	X	X
design e sintassi	X	X	X
factoring	X	X	X
locality	X	X	X
supporto all'astrazione		X	X
espressività		X	X
type checking			X
gestione delle eccezioni			X
restrizione dell'aliasing			X

Programmazione imperativa e strutturata

IBM 704 e Fortran

- Indicizzazione e operazioni floating point in hardware
- FORmula TRANslating system: progettato ad-hoc per questa macchina *e viceversa* (Backus '54)
- primo linguaggio ad alto livello veramente compilato
- obj 1: efficienza del codice macchina scritto a mano
- obj 2: facilità di programmazione degli pseudo-codici
- avrebbe azzerato errori di programmazione e tempi di debugging

IBM 704 e Fortran

- Indicizzazione e operazioni floating point in hardware
- FORmula TRANslating system: progettato ad-hoc per questa macchina *e viceversa* (Backus '54)
- primo linguaggio ad alto livello veramente compilato
- obj 1: efficienza del codice macchina scritto a mano
- obj 2: facilità di programmazione degli pseudo-codici
- avrebbe azzerato errori di programmazione e tempi di debugging
- (... nelle intenzioni dei realizzatori!)

IBM 704 e Fortran

- computers poco potenti, lenti ed inaffidabili
- principalmente utilizzati per calcolo scientifico
- nessun modo efficiente di programmare
- alto costo dei computers rispetto al costo dei programmatori
- obiettivo del compilatore: produrre codice veloce
- includeva *anche* semplici controlli sintattici
- 18 anni-uomo di sviluppo

Fortran I

- If (espressione aritmetica) N_-, N_0, N_+
- Do (etichetta dell'ultima istruzione) variabile = primo_val, ultimo_val
- (hanno corrispondenti istruzioni atomiche su 704)
- nessun istruzione per data-typing
  - variabili che iniziano in I, J, K, L, M, N considerati interi
  - le altre floating.
- il codice deve essere scritto secondo un *formato fisso* (schede perforate) (es. etichette nei primi 5 caratteri della riga, istruzioni non prima del settimo carattere)
- tipo e dimensioni delle variabili dichiarati in modo statico a compile-time
- (di conseguenza) niente ricorsione
- discreto successo di pubblico ed in termini di efficienza

Fortran II

- Bug fixing
- compilazione separata delle subroutines

Fortran IV – 95

IV (o 66) dichiarazioni di tipo, if logico, sottoprogrammi come parametri

77 gestione stringhe, If con Else opzionale

90 punto di rottura

- collezione built-in di funzioni per operare su array (Dotproduct, Matmul, Transpose ...)
- puntatori e array gestiti dinamicamente (Allocatable)
- record
- Exit (break) e Cycle (continue)
- moduli tipo Ada, con Private e Public
- Rimozione del formato fisso per il codice
- Cambio di convenzione per keywords ed identificatori (da FORTRAN a Fortran).

95 Rimozione di costrutti obsoleti, nuovi costrutti tipo Forall

Algol 60

- seconda metà degli anni '50: proliferazione di linguaggi per singole macchine
- petizione alla ACM (americana) per formare un comitato e creare un linguaggio *universale*
- GAMM (associazione tedesca di matemetica applicata) già al lavoro
- non FORTRAN, perchè proprietà di IBM
- obiettivi:
  - più vicino possibile alla notazione matematica, e leggibile con pochi altri commenti
  - deve poter essere utilizzato per descrivere processi di calcolo, anche in pubblicazioni scientifiche
  - traducibile meccanicamente in linguaggio macchina

Algol 60

- Risultato: International Algorithmic Language, poi ALGOrithmic Language
- radici nel FORTRAN
- Molti compromessi, soprattutto tra americani ed europei, es.
  - espressione => variabile VS variabile := espressione
  - "." VS "," per i decimali
- Nel '60 Naur usa idee di Backus e definisce formalmente la sintassi di ALGOL (BNF).

Algol 60

- Struttura a blocchi
- parametri per valore o per nome
- procedure ricorsive
- stack-dynamic arrays

- popolarissimo nelle pubblicazioni scientifiche
- BNF considerata "difficile"
- troppo flessibile, e difficile da implementare (es. parametri per nome)
- non supportato da IBM

Algol 60

```
comment ALGOL 60 Example Program
 Input: ...
 Output: ...
begin
    integer array intlist [1:99];
    integer listlen, counter, sum;
    real average;

    sum := 0;
    comment No built-in I/O facilities!
    readint(listlen);
    if (listlen > 0) ^ (listlen < 100) then
        begin
            for counter := 1 step 1 until listlen do
                begin
                    readint(intlist[counter]);
                    sum := sum + intlist[counter]
                end
            average := sum / counter;
            printstring(``The average is'');

            printreal(average)
        end
    else
        printstring(``error'');
end
```

Programmazione funzionale

Origini

- Intelligenza artificiale: metodo per elaborare dati simbolici
- Esempio pilota: calcolo delle derivate
- ricorsione ed espressioni condizionali, allocazione dinamica e deallocazione automatica di liste
- FLPL non aveva queste features

Rappresentazione dei dati

- solo atomi e liste
- gli atomi possono essere valori numerici o letterali
- gli elementi di una lista possono essere atomi o liste
- es. (A (B C) D (E (EG)))
- es. ((Dante Alighieri) scrisse (la Divina Commedia))

Calcolo

- Svincolarsi dall'idea di "stato" legata all'architettura alla Von Neumann
- qualsiasi calcolo è l'applicazione di funzioni agli argomenti
- niente assegnamenti né variabili
- niente cicli, solo ricorsione

LISP

- il codice ed i dati hanno *lo stesso formato*
- es. (f x y z) è
  - una lista di 4 elementi se interpretato come dato
  - $f(x, y, z)$ se interpretato come codice

LISP

```
;  LISP Example function
   ( DEFUN equal_lists (lis1 lis2)
      ( COND
         ((ATOM lis1) (EQ lis1 lis2))
         ((ATOM lis1) NIL)
         ( (equal_lists (CAR lis1) (CAR lis2))
           (equal_lists (CDR lis1) (CDR lis2))
         )
         (T NIL)
      )
   )
```

Altri linguaggi funzionali: ML

- strutture dati omogenee
- il tipo di ogni variabile ed espressione può essere determinato a compile-time (strong, static typing)
- i tipi vengono inferiti automaticamente dal contesto di un' espressione (type inference)
- include costrutti imperativi

Programmazione logica:

- Idea: (anni '70)
- Programma = algoritmo + strutture dati (Wirth)
- Algoritmo = "logica" + controllo
- "logica" = calcolo predicativo e motore di inferenza ("resolution", Robinson '65)
- fatti (es. madre(anna, silvia))
- regole (es. nonna(X,Y) :- madre(X,Z), madre(Z,Y))

COBOL: COmmon Business Language

- Fine anni '50, FLOW-MATIC, AIMACO, COMTRAN
- Su mandato del Dipartimento di Difesa U.S.A.
- Il primo a svincolarsi dalla notazione matematica e pensato per un particolare ambito applicativo (Computerizing Business Records)
- molto dettagliato sulle strutture dati
  - DEFINE per macro
  - strutture dati gerarchiche (il primo dopo Plankalkül)
  - nomi fino a 30 caratteri, con "hyphens"
  - dichiarazione esplicita del numero di bit per parte intera e decimale per ogni variabile
- debole nella parte procedurale
  - niente funzioni (solo procedure)
  - le prime versioni non permettevano passaggio di parametri

BASIC: Beginner's All-purpose Symbolic Instruction Code

- linguaggio per studenti diversi da scienze ed ingegneria
- accesso al computer tramite terminali
- obiettivo del sistema:
  - facile da imparare ed utilizzare (per non-scienziati)
  - immissione dati e risposta veloce (per lo svolgimento degli esercizi)
  - accesso libero e privato
  - tempo dell'utente *più importante* del tempo macchina
- primo sistema time-sharing

BASIC: Beginner's All-purpose Symbolic Instruction Code

- solo 14 tipi diversi di comandi
- un singolo tipo di dati (numbers = float)
- storicamente legato ad ambienti visuali e prodotti Microsoft

Altri linguaggi

- SNOBOL: string pattern matching
- SIMULA 67: classi e data abstraction
- ALGOL 68: progettato per l'ortogonalità
- Pascal, Modula 2, Oberon: anche strumenti didattici
- Smalltalk: primo linguaggio Object-Oriented
- (C, ADA, C++, Java, PHP ...)