I linguaggi di programmazione

Moreno Marzolla
Dipartimento di Informatica—Scienza e Ingegneria (DISI)
Università di Bologna
http://www.moreno.marzolla.name/

Ringraziamenti

- prof. Mirko Viroli, Università di Bologna
  - http://mirkoviroli.apice.unibo.it/
- prof. Gianluca Amato, Università di Chieti-Pescara
  - http://www.sci.unich.it/~amato/

Hardware e Software

- Hardware: tutto ciò che ha una consistenza fisica. Monitor, stampante, tastiera, mouse, etc. sono esempi di hardware.
- Software: tutto ciò che non ha consistenza fisica, quindi i programmi (Windows, Linux, Word, ...)

Computer e Programmi

- Un programma per computer è costituito da una sequenza di istruzioni che istruiscono il computer sulle operazioni da svolgere
- Analogia “culinaria”
  - Un programma è l'equivalente di una ricetta.
  - Il computer equivale a un cuoco
  - Così come il computer esegue un programma e ci dà dei risultati sullo schermo o sulla stampante, il cuoco esegue la ricetta per produrre una pietanza

Linguaggio macchina

- L'unico linguaggio di programmazione eseguibile dalla CPU è il linguaggio macchina
  - Ogni istruzione è rappresentata da uno o più byte
- PRO
  - Massima efficienza
- CONTRO
  - Difficile da leggere e scrivere per l'operatore umano
  - CPU di tipo diverso potrebbero avere linguaggi macchina diversi e incompatibili

```
7f 45 4c 46 02 01 01 00 00
00 00 00 00 00 00 00 02 00
3e 00 01 00 00 00 40 04 40
00 00 00 00 00 40 00 00 00
00 00 00 00 88 11 00 00 00
00 00 00 00 00 00 00 40 00
38 00 09 00 40 00 1c 00 1b
00 06 00 00 00 05 00 00 00
40 00 00 00 00 00 00 00 40
00 40 00 00 00 00 00 40 00
40 00 00 00 00 00 f8 01 00
00 00 00 00 00 f8 01 00 00
00 00 00 00 08 00 00 00 00
00 00 00 03 00 00 00 04 00
00 00 00 00 0c 07 00 00 00
00 00 00
```

Linguaggio assembly

- Ogni istruzione del linguaggio macchina è rappresentata da una sequenza di caratteri
- PRO
  - Più leggibile (?) del linguaggio macchina
- CONTRO
  - Ogni tipo di CPU ha il proprio linguaggio assembly
  - Scrivere applicazioni complesse è laborioso

```
xor     %ebp,%ebp
mov     %rdx,%r9
pop     %rsi
mov     %rsp,%rdx
and     $0xfffffffffffffff0,%rsp
push    %rax
push    %rsp
mov     $0x4005c0,%r8
mov     $0x400550,%rcx
mov     $0x40052d,%rdi
callq   400420 <__libc_start_main@plt>
hlt
nopw    0x0(%rax,%rax,1)
mov     $0x601047,%eax
push    %rbp
sub     $0x601040,%rax
cmp     $0xe,%rax
mov     %rsp,%rbp
```

Linguaggi di alto livello

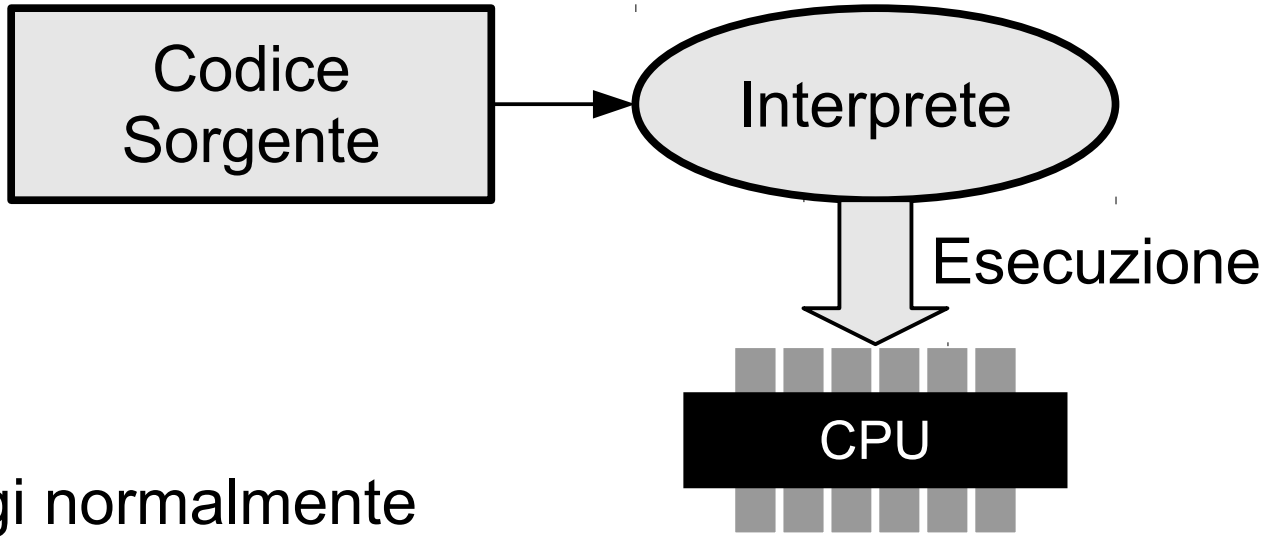
- Consentono di scrivere i programmi ignorando i dettagli della CPU su cui verranno eseguiti
- PRO
  - Semplificano la scrittura di programmi
  - Indipendenti dalla CPU
- CONTRO
  - Non eseguibili direttamente dalla macchina

```
#include <stdio.h>

int main( )
{
  printf("Hello, world\n");
  return 0;
}
```

Interpreti e Compilatori

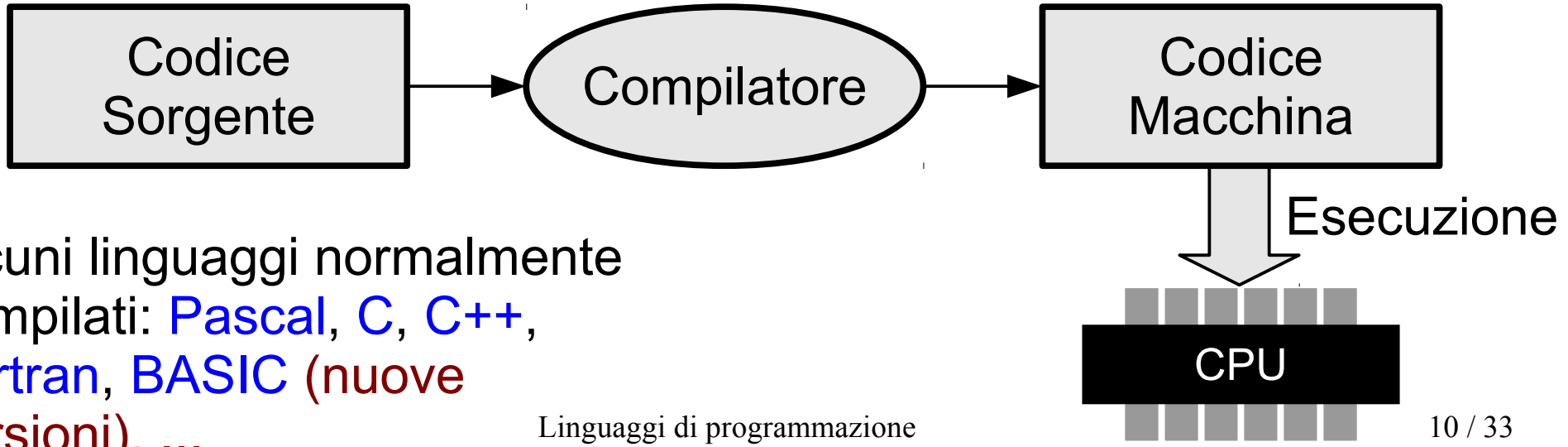
- Interprete
  - Programma che legge una sequenza di istruzioni, espresse in un linguaggio di programmazione di alto livello, e le esegue direttamente



Alcuni linguaggi normalmente interpretati: BASIC (vecchie versioni), Javascript, Perl, Python, ...

Interpreti e Compilatori

- Compilatore
  - Programma che traduce un programma (codice sorgente) scritto in un linguaggio di programmazione di alto livello in un programma equivalente in linguaggio macchina (codice macchina)
  - Il programma in linguaggio macchina può essere direttamente eseguito dalla CPU



Alcuni linguaggi normalmente compilati: Pascal, C, C++, Fortran, BASIC (nuove versioni), ...

Sintassi

- I programmi non possono essere espressi in un linguaggio “naturale” (es., italiano, inglese)
  - Ambiguo
  - Di difficile interpretazione per una macchina
- La sintassi di un linguaggio di programmazione è un insieme di regole che descrivono come deve essere scritto un programma per essere accettato dall'interprete/compilatore
- La sintassi è definita utilizzando una grammatica

Elementi di una grammatica

- Un insieme di simboli (o parole) terminali
  - es: {“il”, “la”, “cane”, “capra”, “gatto”, “mangia”, “dorme”}
- Un insieme di simboli non terminali
  - es: {<frase>, <articolo>, <soggetto>, <verbo>}
- Un insieme di regole di produzione
  - es: { <frase> ::= <articolo> <soggetto> <verbo>
    <articolo> ::= “il” | “la”
    <soggetto> ::= “cane” | “capra” | “gatto”
    <verbo> ::= “mangia” | “dorme” }
  - È possibile usare il simbolo ε a destra di una regola di produzione per indicare la sostituzione vuota
- Un simbolo (non terminale) iniziale
  - es: <frase>

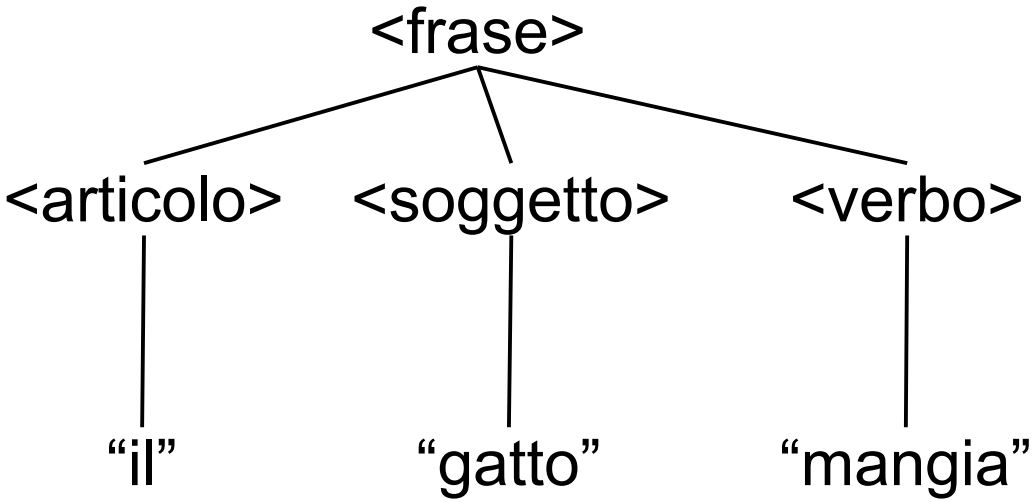
Esempio

- Si parte dal simbolo iniziale (<frase> nel nostro caso)
- Si applica una regola di produzione per sostituire un simbolo non terminale con una delle alternative a destra di ::=
- Il procedimento termina quando si ottengono tutti simboli terminali
  - <frase> → <articolo> <soggetto> <verbo>
    → “il” <soggetto> <verbo>
    → “il” <soggetto> “mangia”
    → “il” “gatto” “mangia”

```
<frase> ::= <articolo> <soggetto> <verbo>
<articolo> ::= “il” | “la”
<soggetto> ::= “cane” | “capra” | “gatto”
<verbo> ::= “mangia” | “dorme”
```

Esempio

- Il processo di derivazione (trasformazione del simbolo iniziale in una frase valida del linguaggio) può essere visualizzato tramite una struttura ad albero




```
<frase> ::= <articolo> <soggetto> <verbo>
<articolo> ::= “il” | “la”
<soggetto> ::= “cane” | “capra” | “gatto”
<verbo> ::= “mangia” | “dorme”
```

Attenzione

- Non tutti i programmi sintatticamente corretti hanno senso
  - Nell'esempio precedente, la frase “la gatto dorme” viene riconosciuta come valida!

Esempio

- Consideriamo la grammatica seguente, con simbolo iniziale <A> e simboli terminali (e)

  <A> ::= ε | (<A>) | <A> <A>

  Sostituzione vuota

- Disegnare l'albero di derivazione per le seguenti stringhe del linguaggio
  - `((()))`
  - `(())()`
  - `()()()`

Esempio

Scrivere una grammatica BNF in grado di generare tutte e sole le sigle presenti sulla rosa dei venti seguente

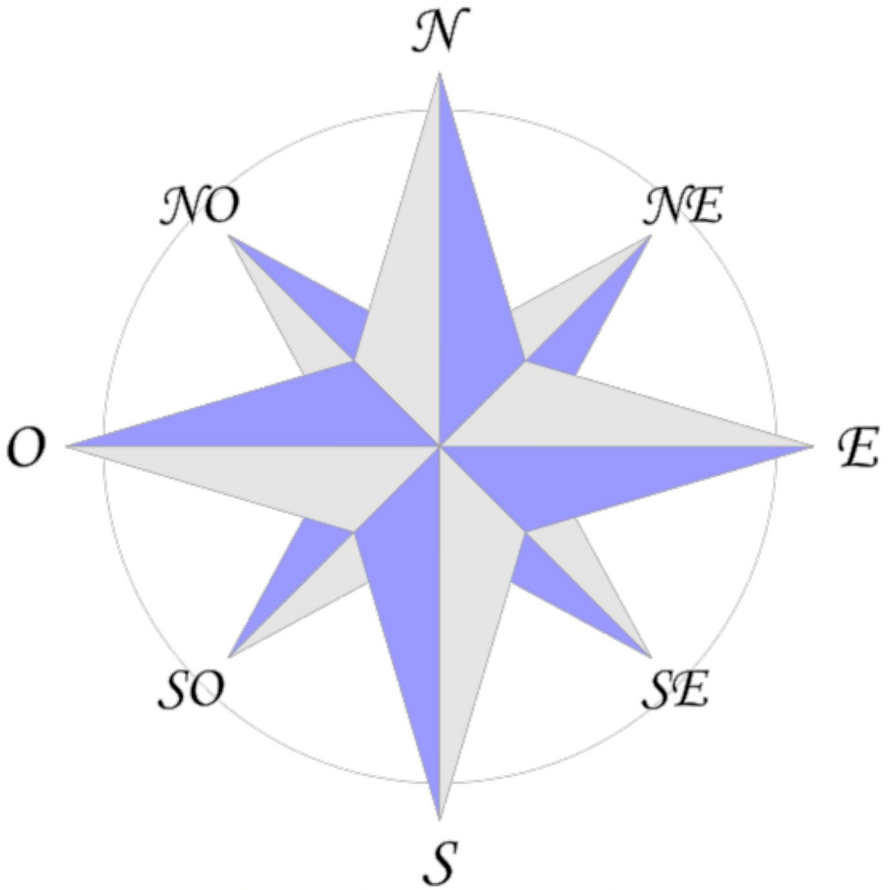


Immagine: https://commons.wikimedia.org/w/index.php?curid=15810334

Esempio

- Definire una grammatica per rappresentare numeri interi senza segno
  - Esempi: 1, 130, 79, 0
- Non sono ammessi numeri con zeri iniziali ridondanti
  - Es, non è ammesso 0023
- Suggerimento: può essere utile partire da una grammatica più semplice che genera anche numeri con zeri iniziali ridondanti. Es:
  - <Num> ::= <Cifra> | <Cifra> <Num>
    <Cifra> ::= "0" | "1" | … | "9"

Esercizio per casa (complesso)

- Definire una grammatica per rappresentare i numeri interi con segno. La grammatica deve essere in grado di generare, ad esempio:
  - -129
  - +79
  - 76
  - 0
- Non sono ammessi zeri ridondanti (no 0023)
- Non deve generare +0 e -0, ma solo 0 (senza segno)

Analisi sintattica

- In realtà, i compilatori (e gli interpreti) effettuano l'operazione inversa
  - Determinano se esiste una sequenza di sostituzioni che, partendo dal simbolo iniziale della grammatica, produca la sequenza di parole presenti nel programma
  - Se esiste, il programma si dice sintatticamente corretto
  - Altrimenti, il compilatore segnala un errore di sintassi

```
#include <stdio.h>
int main( void )
{
    printf("Hello, world!\n")
    return 0;
}
```

```
hello.c: In function ‘main’:
hello.c:5:2: error: expected ‘;’
before ‘return’
  return 0;
  ^
```

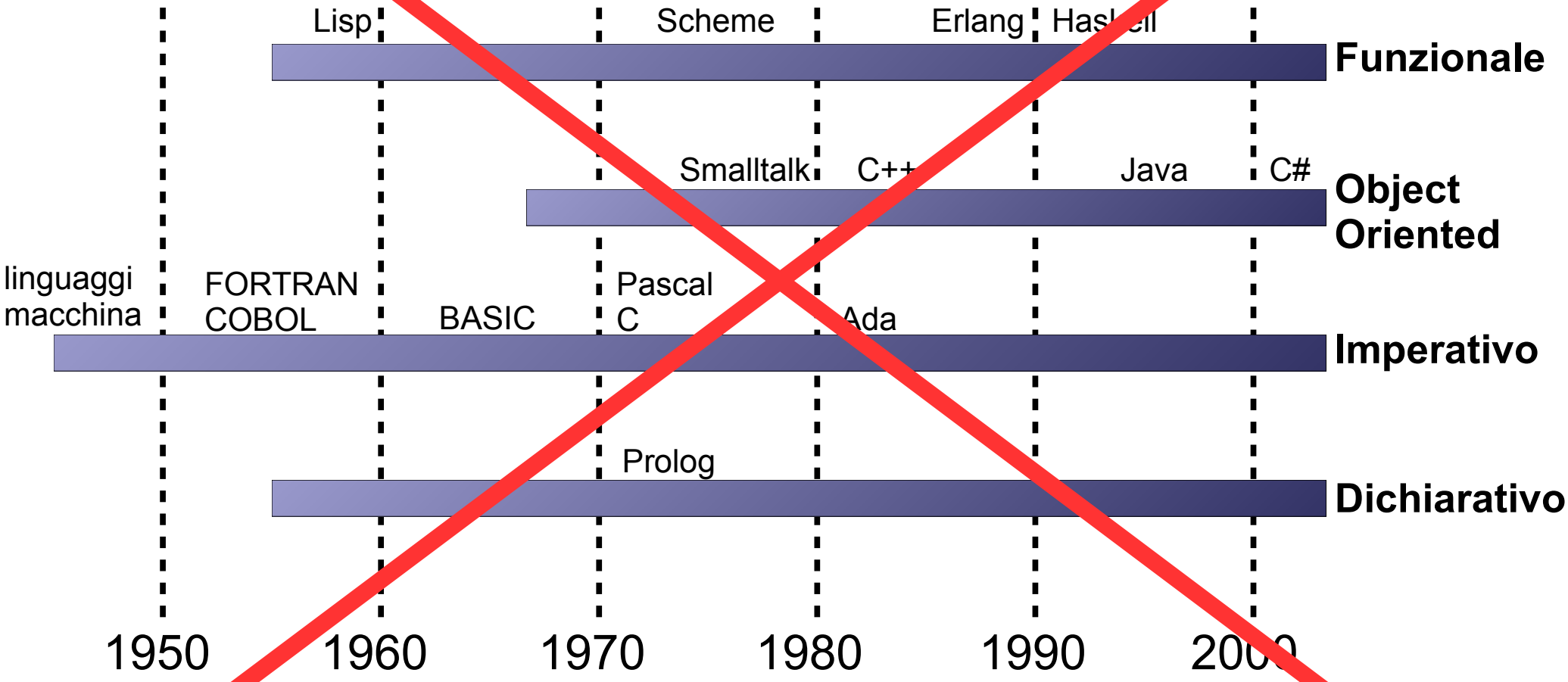
Sintassi e Semantica

Un linguaggio di programmazione è definito:

- Specificando una grammatica
  - Sintassi del linguaggio
- Descrivendo l'effetto dell'esecuzione di un programma
  - Semantica del linguaggio
  - La semantica di un linguaggio definisce il significato di un programma sintatticamente corretto
    - Es: molti linguaggi di programmazione riconoscono come sintatticamente corrette espressioni del tipo `(a + b * 3)`
    - Pero' alcuni linguaggi considerano espressioni del tipo sopra non semanticamente corrette se le variabili a e b non contengono valori numerici (ma contengono, ad es., sequenze di caratteri)

Linguaggi di programmazione

Evoluzione di alcuni linguaggi di programmazione



. Glenn Brookshear, "*Informatica una panoramica generale*", 2004
https://en.wikipedia.org/wiki/Timeline_of_programming_languages

Linguaggi imperativi

- C, Fortran, COBOL, BASIC
- I programmi vengono descritti mediante sequenze di istruzioni che modificano lo stato della computazione

```
int MCD( int n, int m )
{
    while ( n != m ) {
        if ( n > m ) {
            n = n - m;
        } else {
            m = m - n;
        }
    }
    return n;
}
```

Linguaggi funzionali

- Scheme, ML, Haskell, Erlang
- I programmi sono descritti come composizione di funzioni che trasformano input in output
- Le funzioni possono essere passate come parametri, o ritornate come risultato da altre funzioni
  - Molti linguaggi non funzionali non consentono ciò
- I linguaggi funzionali vengono utilizzati in ambiti particolari, richiedono esperienza per essere usati correttamente

```
(define (square x)
    (* x x))

(define (double x)
    (* 2 x))

(define (compose f g)
    (lambda (x) (f (g x))))

((compose square double) 3)
=> 36
```

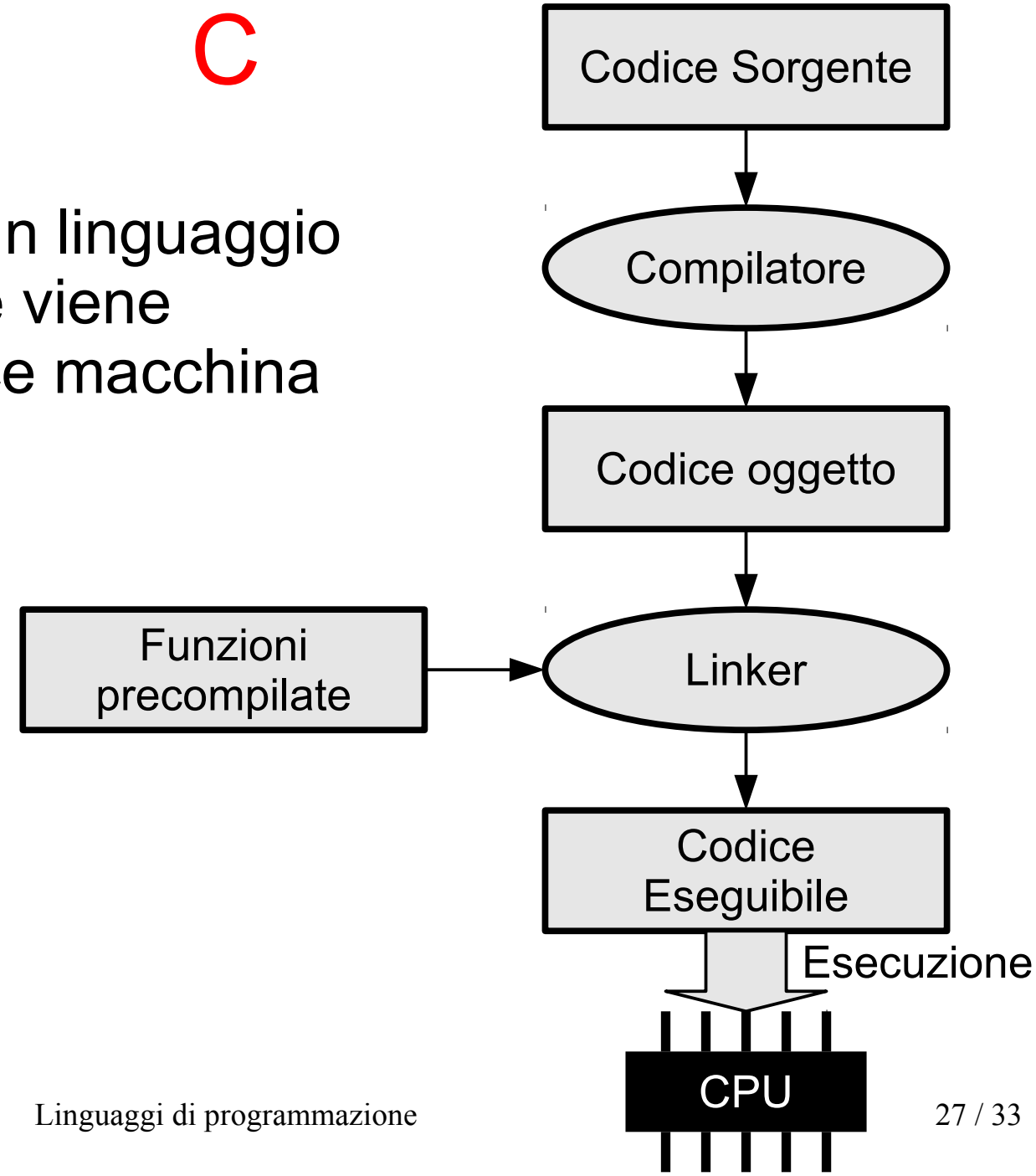
Linguaggi dichiarativi

- Prolog e sue varianti
- Descrivono la logica del programma, anziché le istruzioni da eseguire per calcolare il risultato
- Molto usati nell'ambito dell'Intelligenza Artificiale

```
                              Regole
discendente(X, Y) :-
    genitore(Y, X).
discendente(X, Y) :-
    genitore(Z, X),
    discendente(Z, Y).
                               Fatti
genitore(antonio, carlo).
genitore(fabio, anna).
genitore(anna, andrea).
genitore(anna, giulia).
Chi sono i discendenti di
                              fabio?
?- discendente(X, fabio).
X = anna ? ;
X = andrea ? ;
X = giulia ? ;
no
```

C

- Il linguaggio C è un linguaggio di “alto” livello che viene compilato in codice macchina



Perché C?

- Il linguaggio C fornisce costrutti di programmazione semplici
- Benefici:
  - Facile da imparare
  - Efficiente, quasi come programmare direttamente in assembler
  - Ubiquo: si usa per programmare dai microcontrollori ai supercomputer
  - Ottimi compilatori liberamente disponibili
- Alcuni aspetti del linguaggio richiedono attenzione
  - Es., puntatori e gestione esplicita della memoria

Top Programming Languages 2015

Lucido 2015 / 2016

Language Rank	Types	Spectrum ranking 2015
1. Java		100.0
2. C		99.7
3. C++		99.2
4. Python		95.8
5. C#		91.5
6. PHP		84.3
7. R		83.7
8. JavaScript		82.8
9. Ruby		74.9
10. Matlab		73.3

Top Programming Languages 2016

Lucido 2016 / 2017

Language Rank	Types	Spectrum ranking 2016
1. C		100.0
2. Java		98.1
3. Python		97.9
4. C++		95.8
5. R		87.7
6. C#		86.4
7. PHP		82.4
8. JavaScript		81.9
9. Ruby		74.0
10. Go		71.5

Top Programming Languages 2017

Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C		100.0
3. Java		99.4
4. C++		96.9
5. C#		88.6
6. R		88.1
7. JavaScript		85.3
8. PHP		81.1
9. Go		75.7
10. Swift		74.3

Fonte: http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2017

“Il linguaggio di programmazione che userete maggiormente nella vostra vita lavorativa non è ancora stato inventato”

Concetti chiave

- Hardware vs Software
- Interpreti vs Compilatori
- Grammatiche formali per definire la sintassi dei linguaggi di programmazione
- Cos'è il linguaggio C