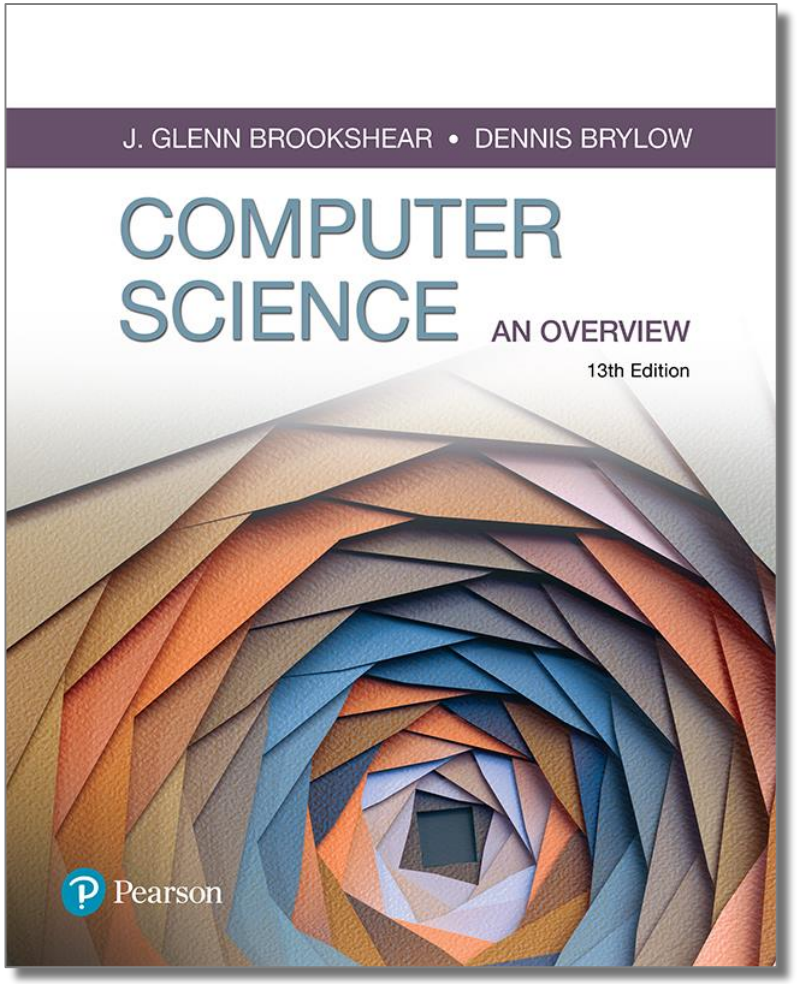
Computer Science An Overview

13th Edition



Chapter 6

Programming Languages

Prospettiva storica

- Prime generazioni
  - Linguaggio Macchina (ex. Vole)
  - Assembly Language
- Linguaggio indipendente dalla macchina
- Estensioni del linguaggio

Seconda generazione: linguaggio Assembly

- Sistema mnemonico per rappresentare istruzioni macchina
  - Nomi mnemomnici per op-codes
  - **Variabili** di programma o **identificatori**: nomi descrittivi per locazioni di memoria, scelti dal programmatore

Caratteristiche del linguaggio Assembly

- Corrispondenza uno-a-uno tra istruzioni ed istruzioni assembly
  - Il programmatore deve pensare come la macchina
- Intrinsecamente dipendente dalla macchina
- Convertito in linguaggio macchina da un programma chiamato **assembler**

Svantaggi del linguaggio Assembly

- Un programma scritto in Assembly non può essere trasportato con facilità da una macchina all'altra.

- Il programmatore è indotto a ragionare secondo i piccoli passi incrementali del linguaggio macchina.
  - Progetto di una casa:
    - Tavole di legno, chiodi, mattoni, …

Esempio di programma

Machine language	Assembly language
156C	LD R5, Prezzo
166D	LD R6, SpeseSpedizione
5056	ADDI R0, R5 R6
30CE	ST R0, CostoTotale
C000	HLT

Linguaggi di terza generazione

- Utilizzano primitive di alto livello (indipendenti dalla macchina)
  - Simile allo pseudocodice nel Cap. 5
- Indipendenti dalla macchina
- Esempi: FORTRAN, COBOL
- Ogni primitiva corrispende ad una sequenza di istruzioni in linguaggio macchina
- Convertito in linguaggio macchina da un programma chiamato **compilatore**

Compilatori ed interpreti

- **Compilatore: programma traduttore** che traduce i programmi scritti con primitive di alto livello in programmi in linguaggio macchina

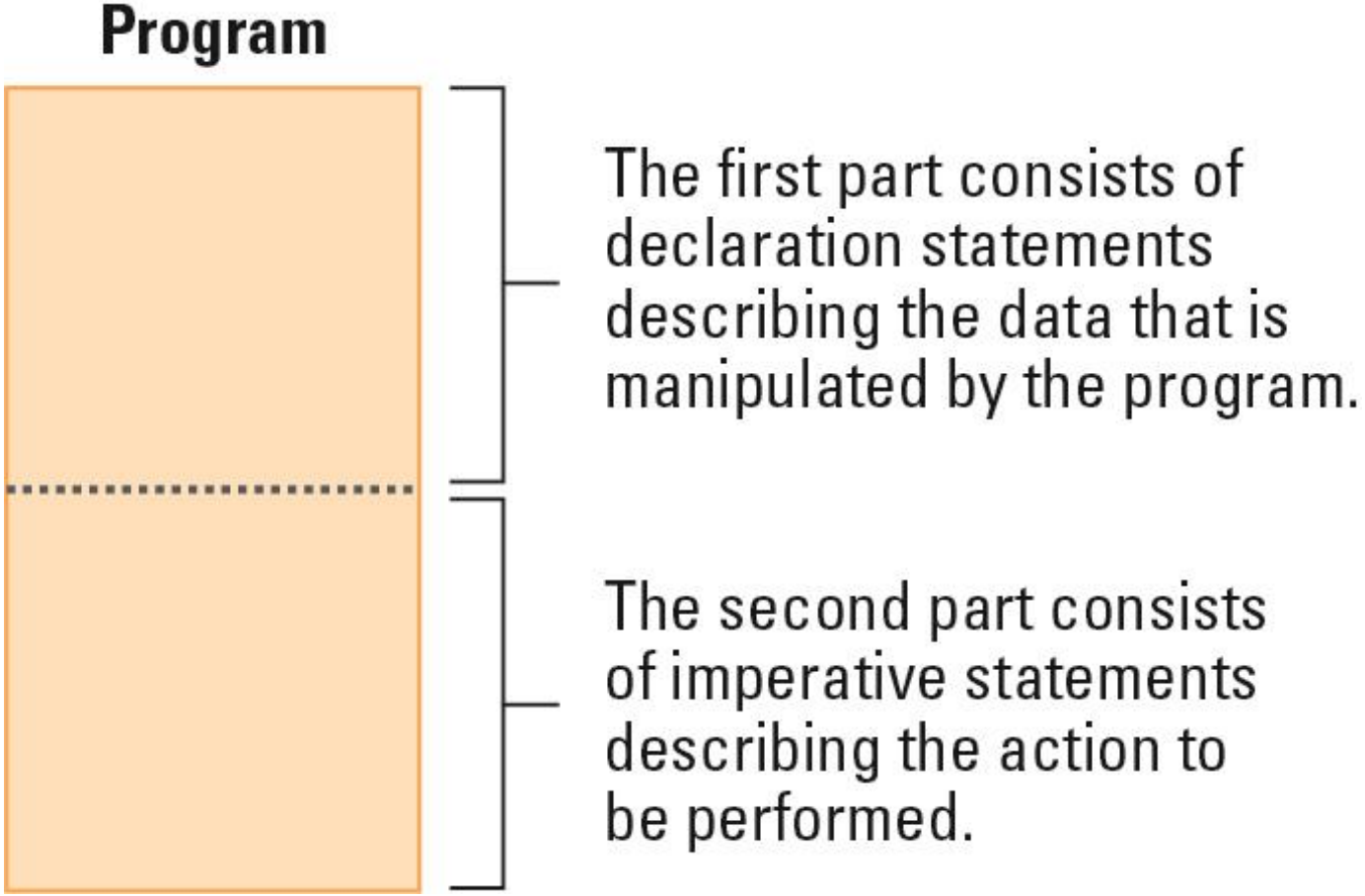
Alternativa: **Interprete.** Simile al traduttore con la differenza che eseguono le istruzioni non appena vengono tradotte.

- esegue le istruzioni man mano che vengono tradotte

Concetti della programmazione tradizionale

- Linguaggi ad alto livello (C, C++, Java, C#, FORTRAN) includono molti tipi di astrazioni
  - Simple: constants, literals, variables
  - Complex: statements, expressions, control

Composizione di un tipico programma



Tipi di dati

- Integer: numeri interi
- Real (float): numeri con frazioni
- Character: Symbols
- Boolean: True/false

Variabili e tipi di dati

```
float Length, Width;
int   Price, Total, Tax;
char  Symbol;

int   WeightLimit = 100;
```

Strutture dati

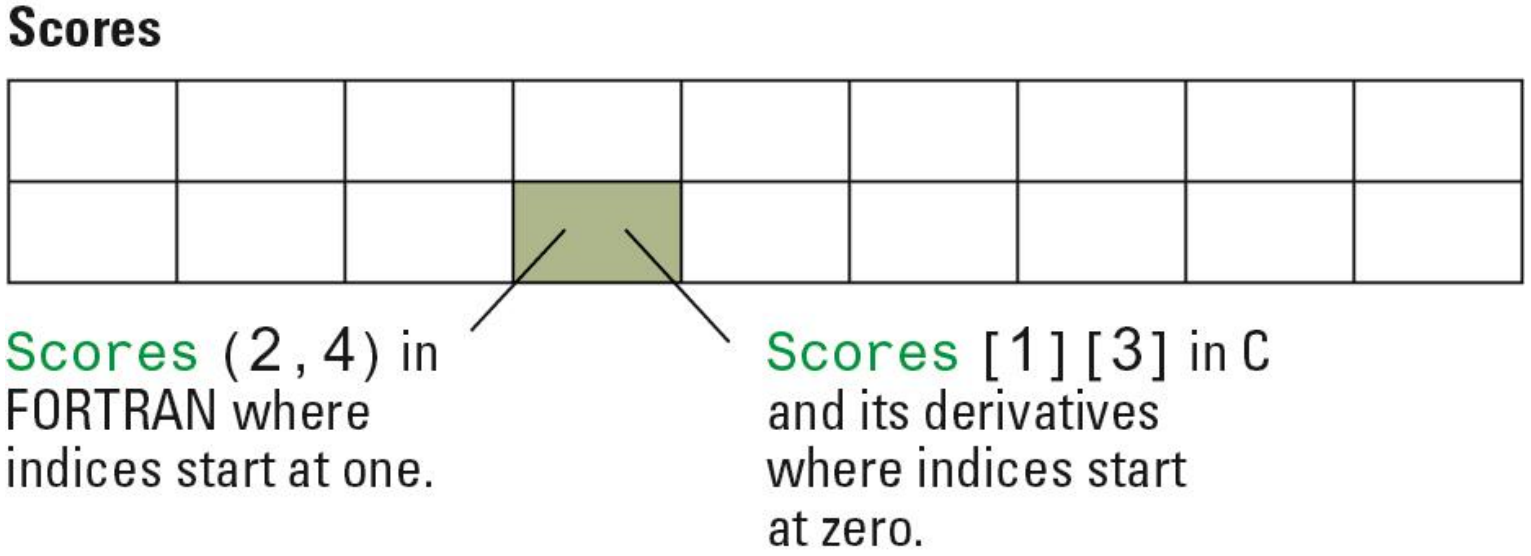
- Arrangiamento di dati
- Struttura commune: **array**
  - In C

```
int Scores[2][9];
```

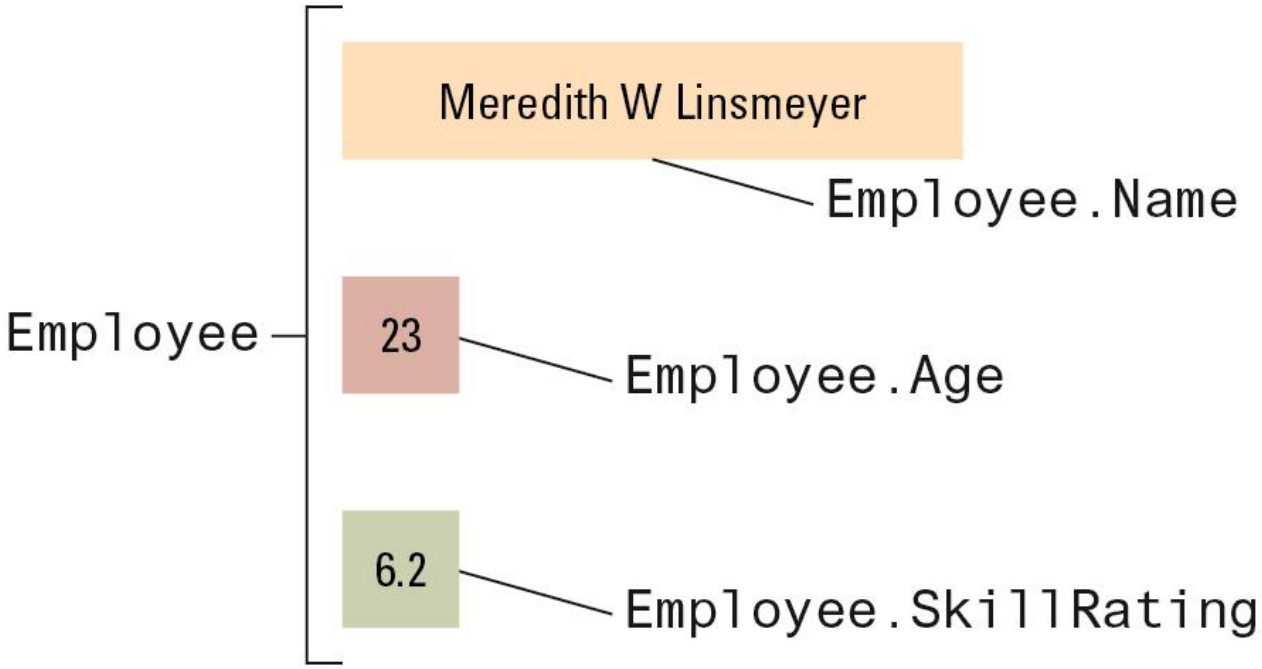
  - In FORTRAN

```
INTEGER Scores(2,9)
```

Array bi-dimensionale (2x3)



Strutture dati




```
struct {    char  Name[25];
            int   Age;
            float SkillRating;
          } Employee;
```

Istruzioni di assegnazione

- In C, C++, C#, Java

```
Z = X + y;
```

- In Ada

```
Z := X + y;
```

- In APL (A Programming Language)

```
Z ← X + y
```

Istruzioni di controllo

- Go to statement

```
        goto 40
    20  Evade()
        goto 70
    40  if (KryptoniteLevel < LethalDose) then goto
60
        goto 20
    60  RescueDamsel()
    70  ...
```

- As a single statement

```
if (KryptoniteLevel < LethalDose):
    RescueDamsel()
else:
    Evade()
```

Istruzioni di controllo (II)

- If in Python

```
if (condition):
    statementA
else:
    statementB
```

- In C, C++, C#, and Java

```
if (condition) statementA; else statementB;
```

- In Ada

```
IF condition THEN
    statementA;
ELSE
    statementB;
END IF;
```

Istruzioni iterative

- While in Python

```
while (condition):
    body
```

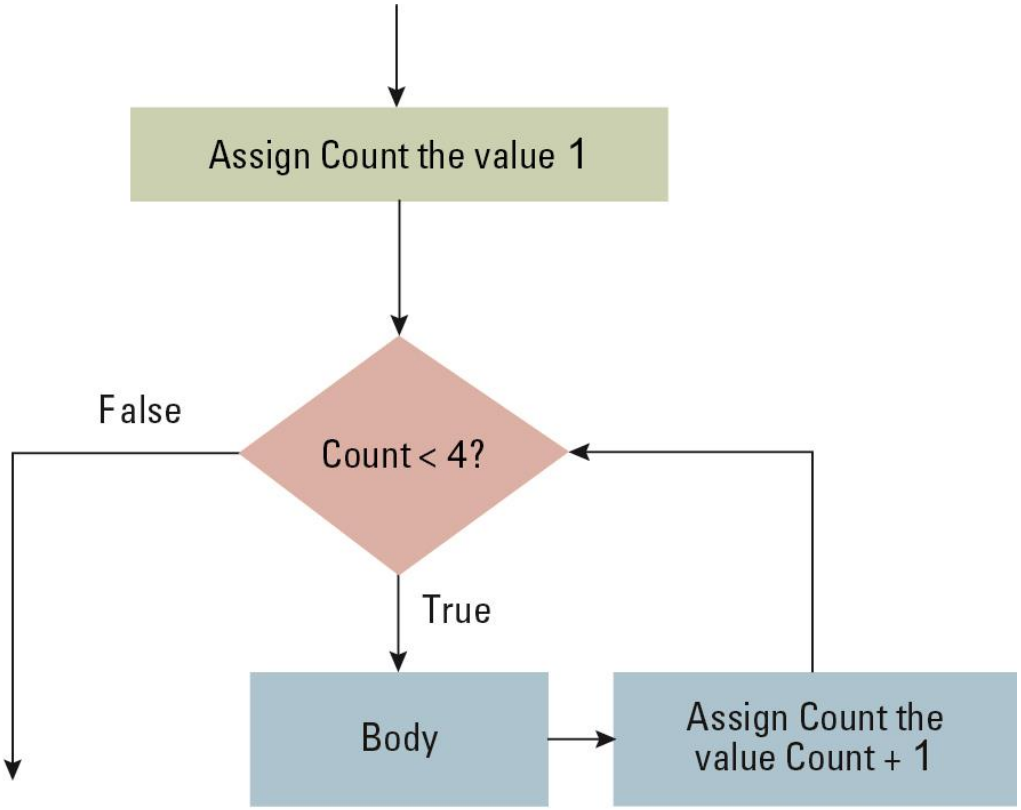
- In C, C++, C#, and Java

```
while (condition)
{ body }
```

- In Ada

```
WHILE condition LOOP
    body
END LOOP;
```

Il ciclo for




```
for (int Count = 1; Count < 4; Count++)
    body;
```

Commenti

- Istruzioni di spiegazione entro in un programma
- Di aiuto nella comprensione del codice
- Ignorati dal compilatore

```
/* This is a comment in C/C++/Java. */

// This is a comment in C/C++/Java.
```

Implementazione del linguaggio

- Il processo di convertire un programma scritto in un linguaggio ad alto livello in una forma eseguibile dalla macchina

Il programma nella sua forma originale è detto **programma sorgente**; la versione tradotta è il **programma oggetto**

Processo di traduzione

- Processo di traduzione:
  - L'analizzatore lessicale riconosce quali stringhe di simboli rappresentano una singola entità o **token**
  - L'analizzatore sintattico (o **parser**) raggruppa i token in statement, usando diagrammi sintattici per creare alberi sintattici.
  - Il generatore di codice costruisce istruzioni in linguaggio macchina per implementare gli statement.

Analizzatore lessicale

- Analisi lessicale: consiste nel riconoscere quali stringhe di simboli del programma sorgente rappresentano singole entità (o **token**)

Esempio:

153 non deve essere intrepretato come 1 5 3, ma come un singolo valore numerico.

pippo non è p i p p o !!!!

Codifica del token

- Analisi lessicale: legge il programma sorgente simbolo per simbolo, identificando quali gruppi di simboli rappresentano dei token.

Classificazione del token con relativa codifica → parser

Compito del parser

- Parser: analizza il programma in termini di unità lessicali (token) anzichè singoli cartatteri.

Classificazione del token con relativa codifica → parser

Il parser identifica la struttura grammaticale del programma

Parole chiave

- La sintassi di un linguaggio deve essere progettata indipendentemente dalla spaziatura fissa usata nel sorgente

Fine di una istruzione: carattere **;**

Parole chiave: non possono essere usate dal programmatore per altri scopi (ad esempio: int, float, if, while, for, …)

Processo di traduzione

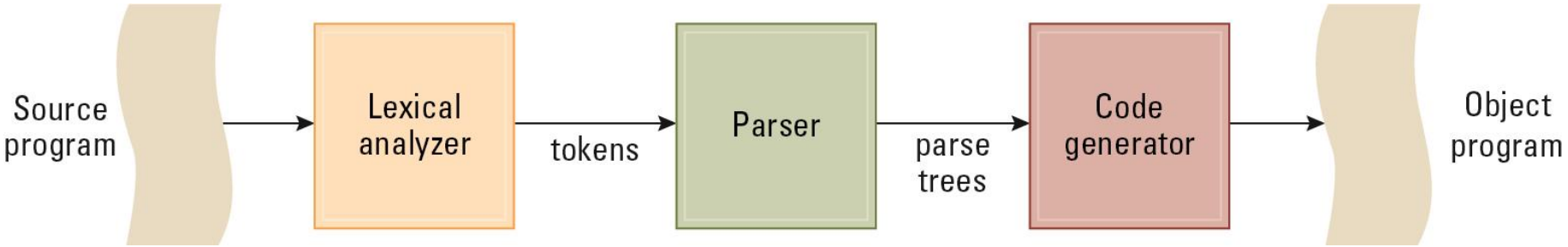


Diagramma sintattico if-else