

# Laboratorio Informatico

Prof. Ing. Daniele Benedetti  
[d.benedetti@alice.it](mailto:d.benedetti@alice.it)

**UNIVERSITA' DEGLI STUDI DI  
MACERATA**

30/10/2012

# Interazione con il docente

- Il docente riceve gli studenti il martedì, dalle 17 alle 19.
- Per ogni comunicazione o avviso, per le dispense e per ulteriori materiali didattici, il docente rimanda gli studenti alla propria pagina personale
- <http://docenti.unimc.it/docenti/daniele-benedetti>
- Il docente è raggiungibile tramite E-Mail [d.benedetti@alice.it](mailto:d.benedetti@alice.it)

# Materiale del corso

- Presentazioni illustrate a lezione
- Libro di testo:
  - Informatica di base 5/ed  
Dennis P. Curtin, Kim Foley, Kunal Sen, Cathleen Morin

# Programma del corso

- Concetti di base dell'ICT
- Architettura degli elaboratori
- **Software: Algoritmi e la programmazione**
- Software di base, i sistemi operativi
- Software applicativi (Office Automation)
- Reti degli Elaboratori

# Il software

- Gli elaboratori sono in grado di svolgere vari compiti grazie a sequenze di istruzioni dette **programmi**
- Il **software** è l'insieme dei programmi che vengono eseguiti dal sistema
- Le nostre attività quotidiane consistono nello svolgimento ordinato di una serie di azioni semplici: cioè di un programma, o **algoritmo**

# Linguaggio naturale vs. linguaggio di programmazione

- Linguaggio naturale:
  - Troppo ambiguo
  - Ingestibile per un computer
  - Incredibile potenza espressiva
- Linguaggio di programmazione:
  - Scarsa espressività
  - Chiaro, coerente e sintetico
  - Potente e comprensibile

# Computer: macchine universali

- Macchine universali: in grado di svolgere una serie pressoché infinita di compiti.
- Per passare da un'operazione ad all'altra, non è necessario modificare i componenti fisici del computer perché questo contiene un dispositivo di memoria (RAM) in cui vengono memorizzate ed eseguite le istruzioni che permettono di svolgere un determinato compito.

# Tipologie di programmi

- **BIOS** (Basic Input Output): sistema fondamentale di input/output incorporato nella memoria non volatile del sistema (ROM). Eseguito al momento dell'accensione del computer.
- **Sistema operativo**: viene eseguito dopo il BIOS, si trova sull'hard disk, ha la funzione di gestire tutte le risorse del computer, mette in funzione i componenti hardware dell'elaboratore
- **Programmi applicativi**: permettono di svolgere dei compiti all'elaboratore, sono memorizzati sull'hard disk, CD-ROM o altre memorie.

# Algoritmo

- Un algoritmo è una soluzione formale ad un problema
- E' una procedura che consta di un numero finito di passi, definiti con precisione, chiaramente comprensibili per l'esecutore, finalizzata a risolvere un problema in maniera automatica
- E' una sequenza di istruzioni che, ricevendo in ingresso un insieme di valori, restituisce in uscita un altro insieme di valori

# Caratteristiche di un algoritmo

- Azioni eseguibili e non ambigue:
  - Non sono ammessi “ un po' ” e “ a piacere ”, che non sono termini adatti ad una macchina
- Determinismo:
  - Fatto un passo, il successivo è uno ed uno solo, ben determinato. Sono possibili alternative ma la scelta è univoca.
- Numero finito di passi.
- Terminazione:
  - L'esecuzione deve finire e produrre un risultato.

# Esempio di algoritmo: prepara il caffè (1)

1. Inizio
2. Svita la moka
3. Riempi d'acqua il serbatoio della moka
4. Inserisci il filtro
5. Riempi il filtro con il caffè
6. Avvita la moka
7. Accendi il fornello

# Esempio di algoritmo: prepara il caffè (2)

8. Attendi
9. Finché il caffè non è pronto, vai al punto 8
10. Spegni il fornello
11. Versa il caffè nella tazzina
12. Se vuoi il caffè amaro vai al punto 15, altrimenti vai al punto 13
13. Metti un cucchiaino di zucchero nella tazzina
14. Finché il caffè è amaro, vai al punto 13.
15. Bevi il caffè
16. Fine.

# Esempio di algoritmo: prepara il caffè (3)

- Questo algoritmo prende in ingresso un insieme di valori (il fornello, la moka, l'acqua, il caffè macinato) e, eseguendo un numero finito di istruzioni, restituisce in uscita un altro insieme di valori (il caffè nella tazzina)

# Esercizio: inviare un sms



# Il prodotto fra due numeri

- Se avessimo un elaboratore in grado di effettuare solo la somma di numeri interi e avessimo la necessità di effettuare operazioni di moltiplicazioni
- Algoritmo per calcolare il prodotto fra due numeri  $a$  e  $b$
- Il prodotto è una somma successiva
- $a * b = a + a + \dots + a \leftarrow (b \text{ volte})$

# Il prodotto: algoritmo

- Con  $a$  e  $b$  in input
- Prodotto = 0
- For  $b$  volte do
  - prodotto = prodotto +  $a$
- End For
- Return prodotto
  
- Esempio  $a = 2$ ,  $b = 5$

# L'iterazione (il ciclo for)

- Una istruzione del tipo for ... do ... end for è detta iterativa o ciclo, in quanto permette di ripetere l'esecuzione di alcune istruzioni per un certo numero di volte
- Esempio: elevamento a potenza

# L'iterazione (il ciclo while)

- Una istruzione del tipo `while ... do ... end` `while` è ancora una volta iterativa, ma differentemente dal ciclo `for`, il numero di volte per cui alcune istruzioni vengono ripetute dipende da una condizione vero/falso
- Il numero di iterazioni non è quindi noto a priori come nel caso del ciclo `for`, ma dipende dal verificarsi della condizione booleana

# L'istruzione condizionale

- Una istruzione del tipo `if ... then ... else ... end if` è detta istruzione condizionale, in quanto regola l'esecuzione dell'algoritmo al verificarsi di determinate condizioni
- **If** *condizione* **then**  
*operazione 1, ... , operazione n*  
**else**  
*operazione 1, ... , operazione n*  
**end if**

# I linguaggi di programmazione

- Sono divisi in generazioni in base al periodo in cui sono stati ideati
- Sono divisi in livelli a seconda della maggiore o minore affinità con il linguaggio del computer, detto linguaggio macchina, i livelli possono essere basso, alto e altissimo; più basso è il livello, più il linguaggio è vicino al linguaggio macchina

# Strutture dati

- I tipi di dato che abbiamo considerato finora (booleano, carattere e numerico) sono detti elementari
- Per convenzione, chiamiamo
  - ♦ bool - booleano
  - ♦ char - carattere
  - ♦ int - numero intero
  - ♦ float - numero a virgola mobile
- ♦ In questo corso, ci limiteremo a trattare come strutture dati: Vettori e Record

# I vettori

- Un array o vettore è una struttura dati omogenea, che contiene un numero finito  $n$  di elementi dello stesso tipo
- Si può immaginare un array come una sorta di casellario, le cui caselle sono dette celle
- Ciascuna delle celle si comporta come una variabile tradizionale; tutte le celle sono variabili di uno stesso tipo, detto tipo base dell'array
- Questi elementi sono individuati attraverso un indice numerico, che tipicamente va da 0 a  $n - 1$

# Esempio di vettore

- Una stringa capace di contenere un nome di una persona
- `char nome[15]`
- |   |   |   |   |   |   |   |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|--|--|--|--|--|--|--|--|
| D | a | n | i | e | l | e |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|--|--|--|--|--|--|--|--|
- Per accedere ad un elemento di un vettore (cioè leggerne il valore o assegnarvi un nuovo valore), utilizziamo l'operatore “[ ]”
- `char a = nome[0]` → qual è il valore di a?
- `nome[1] = 'i'` → cosa succede?
- `nome[3] = 65` → cosa succede?

# I record

- Il record è una struttura dati che può essere eterogenea o omogenea, e quindi può contenere una combinazione di elementi che possono essere di diverso tipo, ad esempio un intero, un numero in virgola mobile e un carattere testuale
- Gli elementi di un record sono detti campi, e sono identificati da un nome

# Esempio

- Esempio: record studente

```
struct studente begin
```

```
    int matricola
```

```
    char nome[20]
```

```
    char cognome[30]
```

```
    boolean fuoriCorso
```

```
    boolean erasmus
```

```
end struct
```

# Accesso ai contenuti di un record

- L'accesso ai campi di un record avviene usando l'operatore “.”

studente s

int m = s.matricola

s.nome = “Giada”

s.erasmus = **false**

# L'insieme delle istruzioni

- Ogni processore è progettato per riconoscere ed interpretare un numero fisso di istruzioni
- Sono istruzioni semplici, come “carica”, “somma”, “salta”
- Ogni processore ha il suo insieme di istruzioni

# I linguaggi macchina: la prima generazione

- Sono costituiti dall'insieme delle istruzioni dei processori, insieme alle regole che specificano come possono essere combinate le istruzioni
- Linguaggi = numero fisso di parole (istruzioni) + regole di come le parole possono essere combinate per formare frasi (sintassi)
- Ogni istruzione è rappresentata internamente da un numero

# Linguaggio macchina: esempio

- 28      1000
- 69      1001
- 52      1002
- 28 = carica
- 69 = somma
- 52 = scrivi
- 1000, 1001, 1002 indirizzo della memoria
- Carica nell'accumulatore il valore nella posizione 1000
- Somma il valore nell'accumulatore con il valore nella posizione 1001
- Scrivi il risultato nella posizione 1002

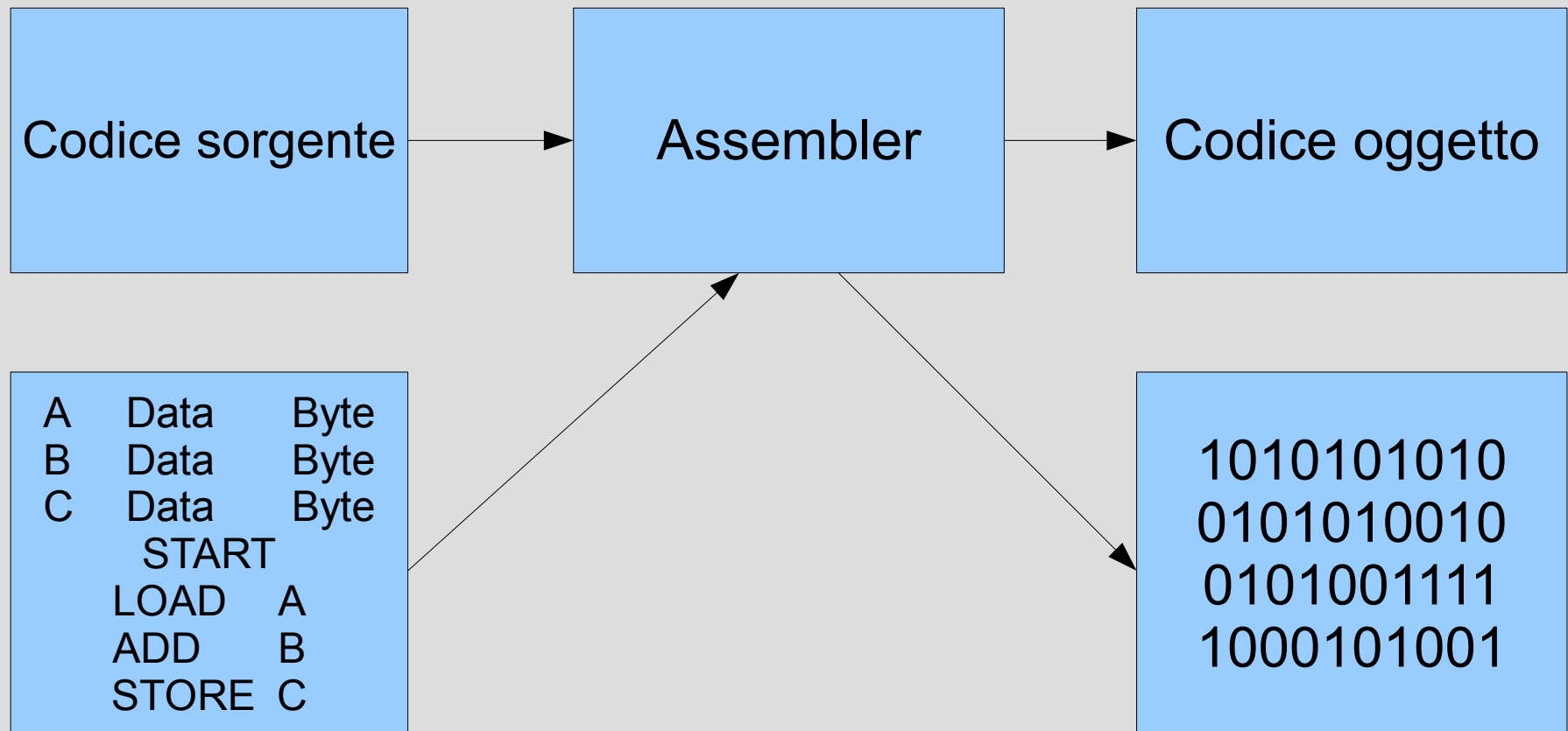
# I linguaggi assemblativi: la seconda generazione

- I linguaggi assemblativi sono specifici di un dato processore
- Ad ogni istruzione numerica viene assegnato un nome convenzionale, mnemonico, più facile da ricordare (28 = LOAD)
- Anche alle posizioni di memoria può essere assegnato un nome
- L'assemblatore traduce il codice sorgente tramite una tabella nell'equivalente codice numerico del linguaggio macchina formando il codice oggetto, salvato in un altro file

# Linguaggio assemblerativo: esempio

- A DATA BYTE
- B DATA BYTE
- C DATA BYTE
- START
  - LOAD A
  - ADD B
  - STORE C

# Da linguaggio assembleativo a linguaggio macchina



# Il linguaggio procedurale: la terza generazione

- Permette operazioni più complesse
- Il traduttore traduce le operazioni in una sequenza di istruzioni macchina
- Il programmatore continua a descrivere ogni procedura nei minimi dettagli
- Esempi: FORTRAN, COBOL, BASIC, Pascal, C, C++ e Java

# I traduttori

- Analizza il programma (codice sorgente) e lo trascrive in linguaggio macchina (codice oggetto)
- Tipi:
  - **Compileri:** intervengono prima che il programma venga eseguito per produrre una versione in linguaggio macchina completa, che viene memorizzata per poter essere successivamente eseguita
  - **Interpreti:** eseguono la traduzione in progressione, ogni istruzione letta, tradotta e eseguita. Esecuzione più lenta. No codice oggetto, è necessario avere interprete sul proprio computer.

# Lo sviluppo dei programmi

- Prima fase: la definizione del problema
- Seconda fase: la progettazione del programma
- Terza fase: la codifica del programma
- Quarta fase: la verifica dei risultati
- Quinta fase: la documentazione
- Scrivere un programma per calcolare il prezzo da pagare

# La definizione del problema

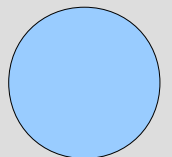
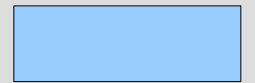
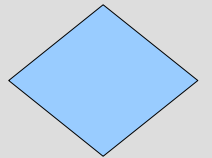
- Definire in maniera precisa ed accurata il problema che si intende risolvere, chiedendo ai clienti o utenti finali, di descrivere l'output atteso del programma
- Esempio: calcolare l'importo da pagare, dati la quantità e l'importo unitario e l'aliquota IVA, sapendo che si applica uno sconto S1 (anch'esso dato) se la quantità supera le 100 unità, altrimenti uno sconto S2 (sempre in input).

# La progettazione del programma

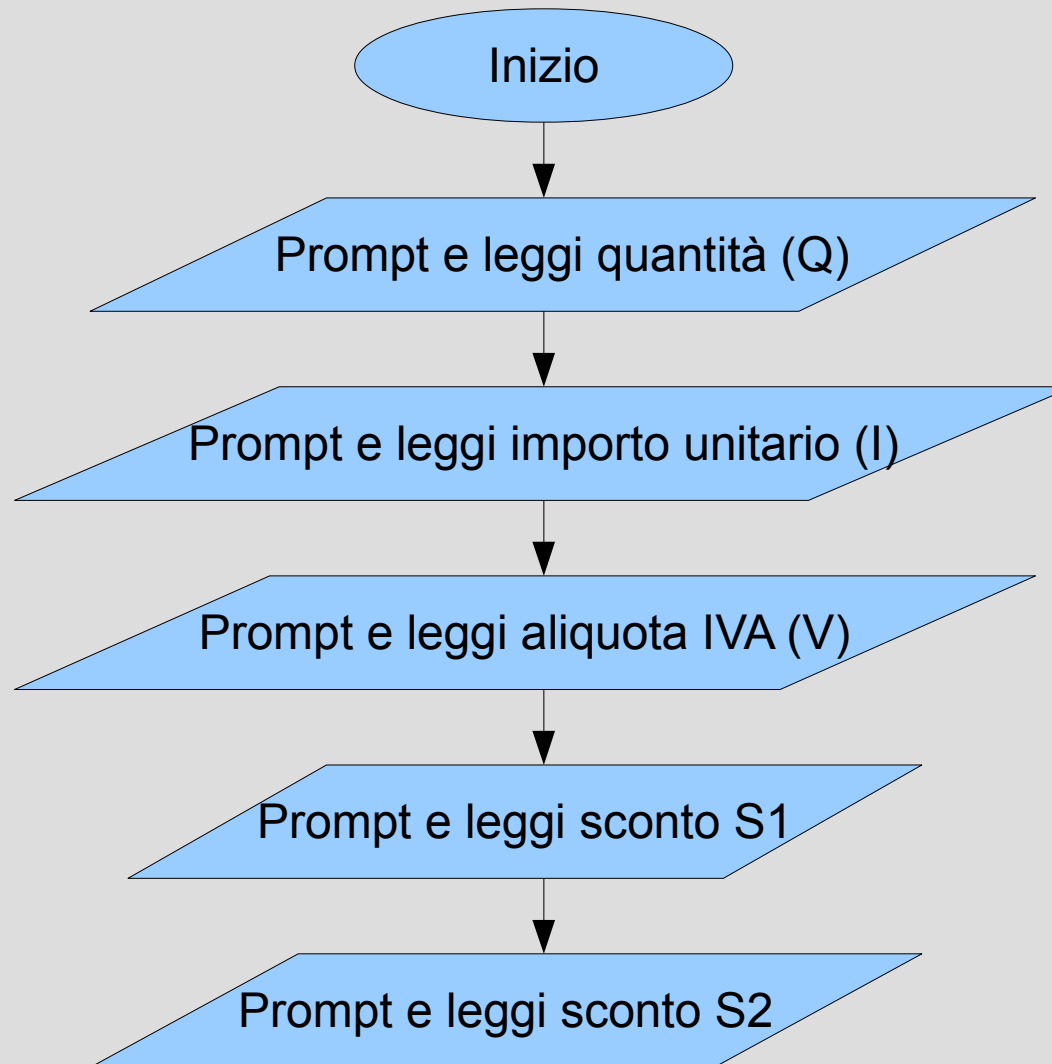
- Obiettivi da raggiungere → Ricerca della soluzione più elegante
- Per scambiarsi idee e descrivere i progetti si usano diverse tecniche, tra cui i **diagrammi di flussi** e gli **pseudocodici**

# I diagrammi di flusso

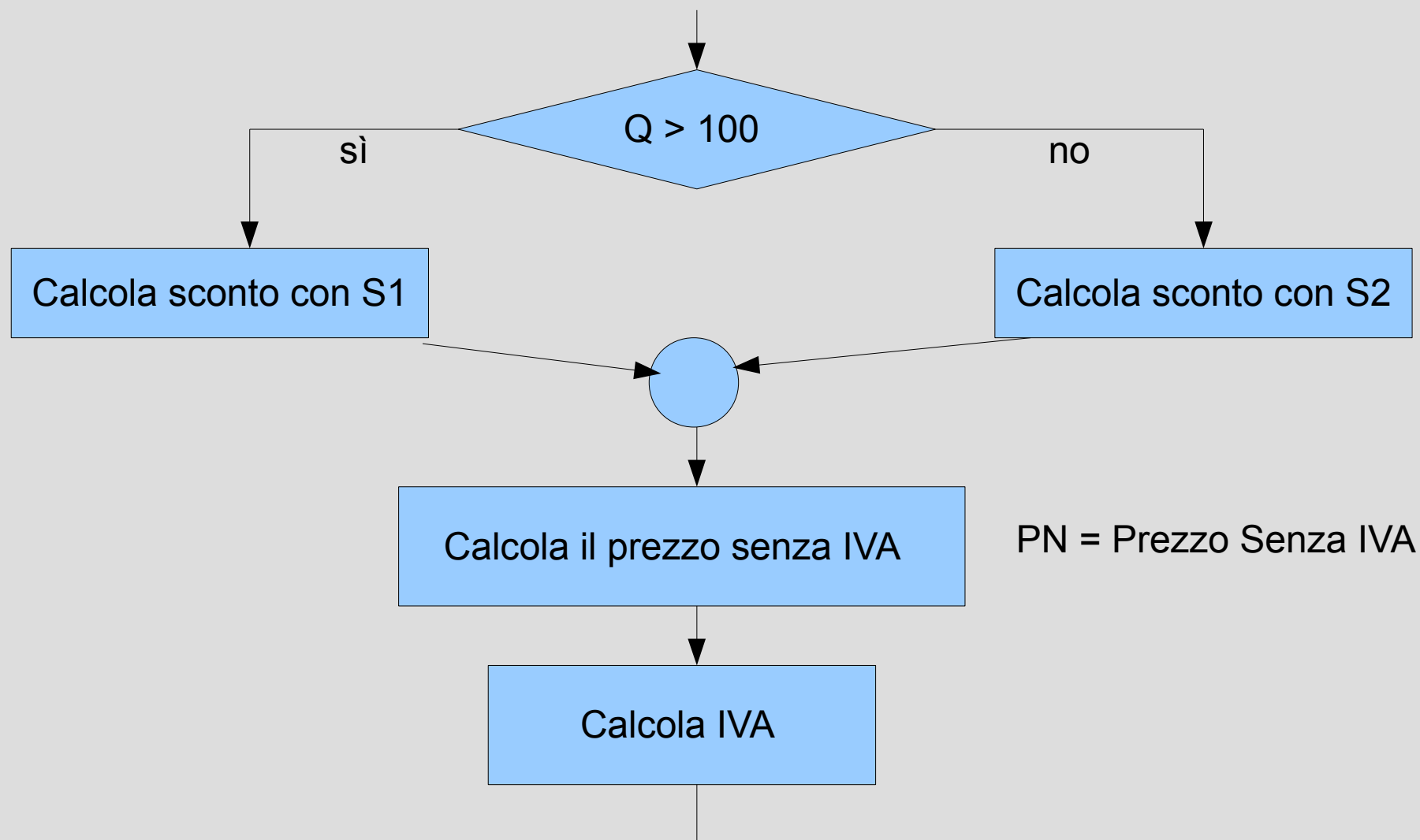
- Grafico per illustrare un processo in tutte le sue fasi
- Le fasi rappresentate con simboli e collegate da linee di flusso
- Il simbolo di terminazione (ovale): inizio o fine
- Il simbolo di decisione (rombo): scelte, if-then-else
- Il simbolo di elaborazione (rettangolo): esecuzione di un calcolo
- Il simbolo di input/output (parallelogramma): immissione o emissione di informazioni
- Il simbolo di connessione (cerchio): unisce due rami del programma



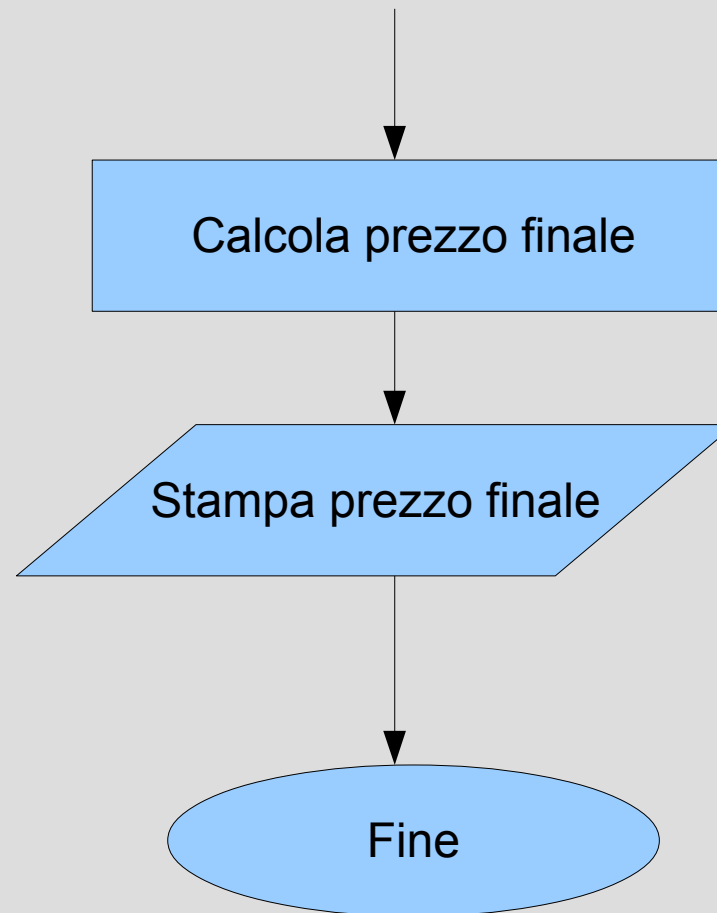
# Diagramma di flusso: esempio (1)



# Diagramma di flusso: esempio (2)



# Diagramma di flusso: esempio (3)



# Gli pseudocodici

- Forma di scrittura del codice affine alla lingua parlata dal programmatore, è simile alla scaletta che ci si prepara nella scrittura di una relazione
- Esempio:

```
Leggi la quantità Q
Leggi l'importo unitario I
Leggi lo sconto S1
Leggi lo sconto S2
Leggi l'aliquotà IVA V
Se  $Q > 100$ 
    Sconto =  $Q * I * S1$ 
Altrimenti
    Sconto =  $Q * I * S2$ 
 $PN = Q * I - \text{Sconto}$ 
 $IVA = PN * V$ 
Prezzo =  $PN + IVA$ 
Stampa Prezzo
```

# Terza fase: la codifica del programma

- Scegliere un linguaggio di programmazione
- Esempio:

```
PRINT "Inserisci quantità"  
INPUT Q  
PRINT "Inserisci importo unitario"  
INPUT I  
PRINT "Inserisci sconto S1"  
INPUT S1  
PRINT "Inserisci sconto S2"  
INPUT S2  
PRINT "Inserisci aliquota IVA"  
INPUT V  
IF Q > 100 THEN Sconto = Q * I * S1  
ELSE Sconto = Q * I * S2  
PN = Q * I - Sconto  
IVA = PN * V  
Prezzo = PN + IVA  
PRINT Prezzo  
END
```

# Quarta fase: la verifica dei risultati

- La fase di verifica e messa a punto del programma alla ricerca, localizzazione e correzione degli errori (bug) presenti nel codice
- Esempio: vanno fatte delle prove inserendo diversi tipologie di valori in input per testare la correttezza degli output, in particolare vanno provati i valori più “delicati”

# Quinta fase: la documentazione

- Scrittura della documentazione (manuale, guida on-line)
- Supporto tecnico usa la documentazione per installare il programma
- I programmatori fanno riferimento alla documentazione per le revisioni e aggiornamenti

# Lo sviluppo dei sistemi in ambito aziendale

1. **Analisi:** analizzare esigenze dell'azienda e definire problema nei dettagli con linguaggi formali e grafici per la modellizzazione dei sistemi (UML)
2. **Progettazione:** scrittura del progetto, coinvolge molte persone, la dirigenza approva il rapporto
3. **Sviluppo:** acquisizione hardware e software, scrittura (eventuale) di programmi personalizzati e verifica del nuovo sistema
4. **Manutenzione:** dopo l'installazione, individuati e introdotti cambiamenti per migliorare il sistema

# Tecniche di programmazione

- I programmi possono essere suddivisi in sezioni in modo che i compiti complessi vengano eseguiti attraverso una sequenza di passi più semplici
- Funzioni
- Sottoprogrammi (subroutine)
- Possiamo usare funzioni e sottoprogrammi per evitare di ripetere delle istruzioni ricorrenti. Possiamo ad esempio usare funzioni standard o scritte da altri per aumentare l'efficienza.

# Esercizi:

- Dati due numeri interi, calcolare e stampare il minore
- Dati due numeri interi, calcolare e stampare la somma
- Dato un numero intero, calcolare se è numero primo e stampare di conseguenza
- Dato in input un array di 10 elementi, stampare gli elementi ordinati in ordine crescente
  - Prendere in input i 10 valori e salvarli in un vettore

# Esercizio per la prossima volta

- Conversione base:
  - Vogliamo un algoritmo che operi la conversione di un numero da una rappresentazione in base  $B$  alla rappresentazione in base 10.