Introduzione alla programmazione

# Le fasi di simulazione e codifica dell’algoritmo:

Quando ci troviamo di fronte a un problema da risolvere con il computer, il primo passo cruciale è ideare un **algoritmo**. Un algoritmo non è altro che una sequenza precisa e finita di istruzioni che, eseguite in un determinato ordine, portano alla soluzione del problema.

Una volta che abbiamo un'idea chiara dell'algoritmo, passiamo attraverso due fasi importanti:

1. **Simulazione (o Prova su carta):** Prima di tradurre l'algoritmo in un linguaggio di programmazione specifico (la codifica vera e propria), è estremamente utile **simularne l'esecuzione**. Questo significa "far finta" di essere il computer ed eseguire manualmente i passaggi dell'algoritmo con alcuni esempi concreti.

**Perché è importante?** La simulazione ci permette di:

* **Verificare la correttezza logica:** Ci assicuriamo che l'algoritmo produca effettivamente il risultato desiderato per diversi input.
* **Individuare errori o "bug" logici:** Possiamo scovare passaggi errati, condizioni mancanti o sequenze illogiche prima di investire tempo nella scrittura del codice.
* **Ottimizzare l'algoritmo:** A volte, durante la simulazione, ci rendiamo conto che alcuni passaggi possono essere eseguiti in modo più efficiente.
* **Comprendere a fondo il funzionamento:** Ci aiuta a interiorizzare la logica dell'algoritmo e a prepararci meglio per la fase di codifica.

**Come si fa?** Prendiamo un foglio di carta e seguiamo passo dopo passo le istruzioni dell'algoritmo, tenendo traccia dei valori delle eventuali variabili coinvolte.

**Esempio: la preparazione del caffè**

Esecutore: essere umano corredato di caffettiera "moka", cucina a gas e macina-caffe`;

**Algoritmo:**

1. svitare la caffettiera;
2. se si dispone di caffè macinato:

* riempire il filtro con il caffè macinato,
* altrimenti se si dispone di caffè in chicchi:
  + macinarlo e ripetere il punto 2;
  + altrimenti terminare (il caffè non si può fare).

1. riempire la parte inferiore della caffettiera con acqua;
2. inserire il filtro nella macchina;
3. avvitare la caffettiera;
4. accendere il fuoco a gas;
5. collocare la moka sul fuoco;
6. attendere l'uscita del caffe`;
7. spegnere il fuoco;
8. fine (il caffè è pronto).

**Esecutore**: essere umano corredato di caffettiera "moka", cucina a gas e macina-caffe`;

**Set di istruzioni:**

• operazioni fondamentali sulla caffettiera:

* svitare
* avvitare
* riempire il filtro
* riempire con acqua
* spostare

• operazioni fondamentali sulla cucina a gas:

* accendere
* spegnere

• operazioni fondamentali sul macina-caffe`:

* macinare

• altre operazioni:

* verifica di condizioni
* ripetizione di operazioni
* attesa
* ..

**Proprietà fondamentali dell’Algoritmo**

**1. Eseguibilità`:** ogni “istruzione” deve essere eseguibile da parte dell’esecutore

dell’algoritmo;

**2. Non Ambiguità`:** ogni istruzione deve essere univocamente interpretabile

dall'esecutore

**3. Finitezza:** il numero totale di azioni da eseguire, per ogni insieme di dati di ingresso,

è finito.

* se almeno una delle 3 proprietà non è soddisfatta, la sequenza non è un algoritmo.

**Altre proprietà desiderabili:**

* **generalità**`: corretto funzionamento dell'algoritmo anche variando alcuni aspetti del problema (ad esempio, la dimensione dell'insieme dei dati, il tipo dei dati, ecc.)
* **efficienza**: tanto minore è il numero di azioni eseguite per la risoluzione del problema, tanto maggiore è l'efficienza.
* **determinismo**: possibilità di prevedere esattamente prima dell'esecuzione la sequenza di azioni che verranno eseguite, per ogni insieme di dati.
* ...

1. **Codifica:** Una volta che siamo ragionevolmente sicuri della correttezza del nostro algoritmo grazie alla simulazione, possiamo passare alla **codifica**. Questa fase consiste nel tradurre l'algoritmo in un linguaggio di programmazione specifico (come Python, Java, C++, ecc.).

**Cosa comporta?**

* **Scrivere il codice sorgente:** Utilizziamo la sintassi e le regole del linguaggio di programmazione scelto per esprimere le istruzioni dell'algoritmo.
* **Utilizzare gli strumenti di sviluppo:** Spesso ci avvaliamo di editor di testo o ambienti di sviluppo integrati (IDE) che facilitano la scrittura, la gestione e l'esecuzione del codice.
* **Rispettare le convenzioni di codifica:** Seguire uno stile di scrittura del codice chiaro e coerente rende il programma più leggibile e manutenibile.

# La programmazione strutturata:

### La **programmazione strutturata** è un paradigma di programmazione nato negli anni '60 come risposta alla complessità crescente dei programmi. L'idea fondamentale è quella di organizzare il codice utilizzando tre strutture di controllo fondamentali:

### **Sequenza:** Le istruzioni vengono eseguite in ordine sequenziale, una dopo l'altra, dall'inizio alla fine. È la struttura più semplice e naturale.

istruzione 1;

istruzione 2;

istruzione 3;

1. **Selezione (o Condizione):** Permette di eseguire blocchi di codice diversi in base al verificarsi o meno di una condizione. Le strutture di selezione più comuni sono:

* **if (se):** Esegue un blocco di codice solo se una determinata condizione è vera.
* **if-else (se-altrimenti):** Esegue un blocco di codice se una condizione è vera, altrimenti ne esegue un altro.
* **if-elif-else (se-altrimenti se-altrimenti):** Permette di verificare più condizioni in sequenza.
* **switch-case (a seconda di):** Utile quando si devono confrontare una variabile con diversi valori costanti.

if (condizione) {

// blocco di codice da eseguire se la condizione è vera

}

if (condizione) {

// blocco di codice se la condizione è vera

}

else

{

// blocco di codice se la condizione è falsa

}

### **Iterazione (o Ciclo):** Permette di ripetere l'esecuzione di un blocco di codice più volte, finché una determinata condizione è vera o finché non si raggiunge un certo numero di ripetizioni. Le strutture di iterazione più comuni sono:

### **while (mentre):** Esegue il blocco di codice finché una condizione specificata rimane vera.

### **for (per):** Esegue il blocco di codice un numero predefinito di volte, spesso utilizzando un contatore.

### **do-while (fai-mentre):** Simile al while, ma la condizione viene verificata alla fine del ciclo, garantendo che il blocco di codice venga eseguito almeno una volta.

while (condizione) {

// blocco di codice da ripetere finché la condizione è vera

}

for (inizializzazione; condizione; incremento) {

// blocco di codice da ripetere

}

## I vantaggi della programmazione strutturata:

* **Maggiore leggibilità:** L'uso di strutture chiare rende il codice più facile da capire e seguire.
* **Minore complessità:** Aiuta a organizzare il codice in blocchi logici, riducendo la complessità generale del programma.
* **Facilità di manutenzione:** Un codice ben strutturato è più facile da modificare e correggere.
* **Riduzione degli errori:** L'approccio strutturato tende a minimizzare l'uso di istruzioni di "salto" incondizionato (come il goto), che possono rendere il flusso del programma difficile da seguire e aumentare il rischio di errori.

## Gli schemi di flusso (o diagrammi di flusso)

Gli schemi di flusso sono rappresentazioni grafiche degli algoritmi. Utilizzano simboli standardizzati per descrivere le diverse operazioni e il flusso di controllo dell'algoritmo.

### **A cosa servono?**

### **Visualizzare la logica:** Rendono più facile comprendere la sequenza di passi e le decisioni all'interno di un algoritmo.

### **Comunicare l'algoritmo:** Sono un modo efficace per condividere la logica di un programma con altre persone (ad esempio, altri sviluppatori o persone non tecniche).

### **Documentare il codice:** Possono essere utilizzati per documentare il funzionamento di un programma.

### **Aiutare nella fase di progettazione:** Costruire un diagramma di flusso prima di scrivere il codice può aiutare a chiarire la logica e a identificare eventuali problemi.

### **Simboli principali:** Alcuni dei simboli più comuni utilizzati negli schemi di flusso includono:

### **Ovale (Terminale):** Indica l'inizio o la fine dell'algoritmo.

### **Rettangolo (Processo):** Rappresenta un'operazione o un'istruzione da eseguire.

### **Rombo (Decisione):** Indica un punto in cui viene valutata una condizione e il flusso del programma si dirama a seconda del risultato (vero o falso).

### **Parallelogramma (Input/Output):** Rappresenta l'operazione di inserimento di dati (input) o di visualizzazione di risultati (output).

### **Freccia (Flusso):** Indica la direzione del flusso di esecuzione, collegando i diversi simboli.

### **Cerchio (Connettore):** Utilizzato per collegare parti diverse dello schema di flusso, specialmente quando diventa troppo grande per essere disegnato su una singola pagina.

### In sintesi, il Modulo 3 introduce i concetti fondamentali per la creazione di programmi: dalla pianificazione logica attraverso gli algoritmi, alla loro verifica tramite la simulazione, alla traduzione in codice seguendo i principi della programmazione strutturata, fino alla rappresentazione grafica degli algoritmi con gli schemi di flusso. Questi sono mattoni essenziali per costruire programmi più complessi e ben organizzati.