CAPITOLO 2
PROGRAMMAZIONE, ALGORITMI E STRUTTURE DATI

SOMMARIO: 1. Linguaggi di programmazione. - 1.1. Algoritmi. - 1.2. Definizione e caratteristiche di un Algoritmo. - 1.3. Algoritmi deterministici e non deterministici. - 1.4. Efficacia ed efficienza di un Algoritmo. - 1.5. Calcolabilità e Computabilità di un Algoritmo. - 1.5.1. Tesi di Church-Turing. - 1.5.2. Tesi di Mitchie. - 1.5.3. Complessità di un Algoritmo. - 1.6. Il problema dell'arresto. - 1.7. Diagrammi di flusso o flow-chart. - 1.7.1. Sequenza, Selezione ed Iterazione. - 1.7.2. Elementi grafici dei Diagrammi di Flusso. - 1.8. Pseudocodifica. - 1.8.1. Sequenza. - 1.8.2. Selezione. - 1.8.3. Iterazione. - 2. Tecniche di sviluppo di un Algoritmo. - 2.1. Sviluppo Top Down. - 2.2. Sviluppo Bottom Up. - 3. Linguaggi di programmazione. - 3.1. Variabili ed istruzioni. - 3.2. Il codice sorgente. - 3.3. Tipi di linguaggi di programmazione. - 3.4. Ciclo di vita del software. - 3.5. Ambienti di sviluppo. - 2. Paradigmi di programmazione. - 3. Analisi e progettazione del software. - 4. Strutture dati elementari. - 4.1. Le strutture dati dal punto di vista logico. - 4.2. Strutture sequenziali. - 4.3. Strutture concatenate. - 4.3.1. Le strutture lineari. - 4.3.2. Le strutture non lineari. - 4.4. Strutture dati dal punto di vista fisico. - 4.5. Array e Record. - 5. Paradigmi algoritmici. - 5.1. Il linguaggio Pascal. - 5.1.1. Dati e Variabili. - 5.1.2. Strutture iterative. - 5.1.3. Strutture di selezione. - 5.1.4. Selezione binaria. - 5.1.5. Selezione multipla. - 5.1.6. Comandi ed Operazioni. - 5.1.7. Gli operatori booleani e i segni di operazione. - 5.2. Il linguaggio C. - 5.2.1. Ambiente di programmazione. - 5.2.2. Elementi di programmazione in C. - 5.2.3. La compilazione. - 5.2.4. L'input/output. - 5.2.5. Le variabili. - 5.2.6. Dati. - 5.2.7. Gli operatori. - 5.2.8. Operatori aritmetici. - 5.2.9. Operatori di confronto. - 5.2.10. Operatori Logici. - 5.2.11. Struttura condizionale semplice e multipla. - 5.2.12. IF – ELSE. - 5.2.13. Switch. - 5.2.14. Strutture Iterative. - 6. Complessità di algoritmi e problemi.

1. Linguaggi di programmazione.

■ 1.1. Algoritmi.

Il termine "*programma" è ormai di uso comune* essendo il computer, lo smartphone e il tablet ormai macchine di uso comune, non intelligenti ma un mero esecutore di ordini e procedure impartite dall'utilizzatore. Un programma necessita però di una serie di istruzioni, scritte in un opportuno codice detto **linguaggio di programmazione**.

Il programmatore deve però prima analizzare il problema da risolvere, definire gli obiettivi da raggiungere, individuare i dati iniziali (input) e finali (output) e descrivere tutti i passi necessari per ottenere il risultato. Alla fine, si passa a codificare le istruzioni del programma in un Linguaggio di programmazione adatto (C, C++, Java, Visual Basic, Pascal, … etc.). È dunque necessario che venga indicato con precisione come risolvere una classe di problemi che differiscono per i dati iniziali. Questo sottintende la creazione di un algoritmo. Il termine **algoritmo** deriva dal nome del matematico arabo Al Khwarismi (IX sec.) che pubblicò il testo

"*Kitab Al-jabrwal Muquabala*" ossia "*L'arte di numerare ed ordinare le parti in tutto*" da cui deriva il nome **algebra**.

■ 1.2. Definizione e caratteristiche di un Algoritmo.

Un **algoritmo** è un numero finito di istruzioni che definisce l'insieme delle azioni da compiere per poter risolvere un problema. Questo numero di istruzioni è necessariamente finito, ordinato, costituito di passi elementari, comprensibili, che determinano un procedimento mirato a risolvere, in un tempo finito, una classe di problemi, utilizzando dati iniziali (input) e ottenendo dei risultati finali (output).

Definiamo ora nel dettaglio le principali caratteristiche di un Algoritmo, esso deve essere

- *generale*: il metodo deve risolvere una classe di problemi e non un singolo problema
- *finito*: il numero delle istruzioni che la compone deve essere finito, come il numero di volte che ogni istruzione deve essere eseguita
- *completo*: deve contemplare tutti i casi possibili del problema da risolvere
- *non ambiguo*: ogni istruzione deve essere implementato in modo univoco, senza possibilità di ambiguità sul significato dell'operazione
- *eseguibile*: deve poter essere eseguita ogni istruzione in un tempo finito.

■ 1.3. Algoritmi deterministici e non deterministici.

Gli algoritmi possono venire classificati in deterministici e non deterministici. In un algoritmo d**eterministico** se a fronte degli stessi dati di partenza (input) produce gli stessi risultati finali (output). Invece in un algoritmo **non deterministico** se produce risultati diversi a partire da uno stesso insieme di dati iniziali.

■ 1.4. Efficacia ed efficienza di un Algoritmo.

Compito di algoritmo è quindi quello di risolvere una determinata classe di problemi. La **correttezza** dell'Algoritmo dipende ovviamente dai risultati finali che devono essere coerenti con le aspettative. Inoltre, l'**efficienza** di un Algoritmo di misura dalla velocità con cui questo raggiunge il risultato atteso.

Si osservi che dato un problema ciascuno potrebbe realizzare a modo proprio una serie di istruzioni per trovarne la soluzione, dando luogo ad ambiguità o ad istruzioni poco comprensibili a tutti gli altri. Per ovviare a questa difficoltà i programmatori hanno sviluppato diverse tecniche di rappresentazione sia grafica che formale di un algoritmo, in modo da evitare le problematiche precedenti.

Esistono algoritmi che ricevuto in input un'istanza di un problema ed una ipotetica soluzione, possono verificare se quella è effettivamente soluzione del problema dato. Questi vengono chiamati appunto **algoritmi di verifica**. Si capisce banalmente che il più delle volte verificare una soluzione è sicuramente più facile che trovarla.