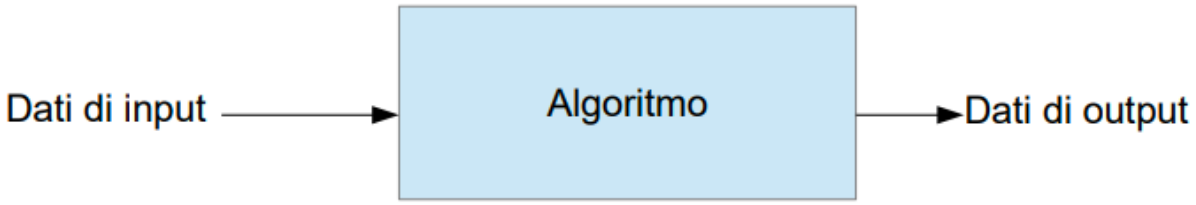
1 - Algoritmi e Programmi

Algoritmo

Un algoritmo è una sequenza di azioni che un esecutore deve compiere per giungere alla soluzione di un qualsiasi problema computazionale, prende in input dei dati e li trasforma per produrre dei dati in output.
Di solito viene rappresentato in una scatola di questo tipo:



L'algoritmo e i dati sono strettamente legati, poichè le operazioni che esegue un algoritmo è condizionato dai dati che riceve

Algoritmi e problemi

Il **problema computazionale** e **l'algoritmo** son due concetti distinti:

- Il **problema computazionale** specifica il risultato che si vuole ottenere, ovvero la relazione che lega i dati di input con i dati di output
- L'**algoritmo** definisce la serie di passi da eseguire per risolvere un problema

Un problema deve essere sempre definito;
Le definizioni di un problema devono essere complete, definire tutti i casi e sopratutto non essere **ambigua**.

Un possibile esempio è quello del minimo:
Il minimo di un insieme A è l'elemento di A che è minore o uguale ad ogni elemento di A, definito matematicamente in:

$$min(A) = a \in A \Leftrightarrow \forall b \in A : a \leq b$$

La soluzione per questo problema è il confronto di ogni elemento con tutti gli altri, quello minore di tutti è il minimo

Descrizione di algoritmi

La descrizione di un algoritmo è composta da azioni elementari

- L'effetto di un’azione su un dato è certo, unico e ripetibile (non dipende da fattori casuali)
- Le azioni devono essere comprensibili e non ambigue

Valutazioni di algoritmi

Ogni algoritmo deve essere valutato in modo che produca il risultato atteso (correttezza), la correttezza di un algoritmo richiede una dimostrazione matematica, in genere **per induzione**

Un algoritmo poi deve essere **efficiente**, sia in termini di spazio (velocità) e spazio (occupazione di memoria)

Oltre a queste principali ci sono altre proprietà importanti

- Robustezza e sicurezza (gestione dei casi limite)
- Semplicità (possibilità di lettura semplificata e di espansione)
- Modularità
- Espandibilità
- Manutenibilità

Progettare un algoritmo

Per poter progettare un algoritmo bisogna:

- **Saper comprendere ed identificare un problema** (nel caso ci sia una soluzione robusta già presente saperla riutilizzare)
- **Saper utilizzare le principali tecniche algoritmiche**

Ciclo di sviluppo del Software

In uno sviluppo del software avvengono diverse operazioni:

- **Studio di fattibilità**: valutare costi e benefici del sistema da costruire
- **Raccolta e analisi dei requisiti**: definire il problema e specificare l'ambiente di sviluppo (sia software che hardware) per creare un **documento di analisi**
- **Progettazione**: individuare la soluzione del problema, andando a creare **il codice**
- **Verifica**: analisi del programma e della sua documentazione tramite prove di correttezza sintattiche e logiche (con test empirici o prove formali, di solito difficili da realizzare in sistemi complessi)
- **Manutenzione**: controllare che il programma durante l'esecuzione produca i risultati attesi e aggiornalo ove necessario

Qualità dei programmi

La fase di progettazione deve fornire un'analisi delle qualità che il programma deve possedere, si suddividono in:

- **Esterne**: caratteristiche evidenziabili dal solo funzionamento del programma durante la fase di esercizio, di solito visibili all'utente
- **Interne**: caratteristiche analizzabili e valutabili da esperti (sviluppatori in questo caso), attraverso uno studio delle scelte tecniche adottate

Qualità esterne

- **Correttezza**: capacità di eseguire precisamente i compiti individuati durante l’analisi dei requisiti
- **Efficienza**: capacità di utilizzare in modo razionale ed economico le risorse di calcolo (in relazione all'obbiettivo)
- **Robustezza**: capacità di funzionare in modo soddisfacente in condizioni limite o anomale rispetto a quelle previste in fase di analisi dei requisiti
- **Usabilità**: capacità di consentire un interazione semplice, naturale ed efficace con l'utente finale

Qualità interne

- **Riusabilità**: capacità di essere riutilizzato, in tutto o in parte, per applicazioni diverse rispetto a quella per la quale è stato prodotto
- **Modularità**: grado di organizzazione interna del programma (strutturazione delle singole parti, della funzionalità e del modo in cui cooperano per l’obiettivo generale)
- **Estensibilità**: capacità di adattarsi facilmente a modifiche nei requisiti
- **Portabilità e Compatibilità**: facilità di trasferire il software prodotto in ambiti diversi
- **Leggibilità**: Capacità del codice di essere autoesplicante
- **Bontà della documentazione**: completezza ed efficacia dei documenti annessi

Principi di Progettazione

Nella progettazione un ruolo fondamentale è giocato dall'**astrazione** e le sue diverse tecniche come:

- Programmazione strutturata
- Modularizzazione
- **Astrazione dati**

Astrazione (nei sistemi software)

L'**astrazione** non è altro che la descrizione di un sistema che pone enfasi su alcuni dettagli o proprietà eliminandone altri (temporaneamente o permanentemente).

Il processo di modellazione è analogo alla modellazione analitica in altri campi: si parte da osservazioni, si formulano ipotesi che spiegano le osservazioni e si usano queste ultime per costruire il modello, mentre le variabili o i parametri possono essere ricavati dagli assiomi o stimati dalle osservazioni.

I requisiti o le funzionalità del sistema giocano il ruolo di osservazioni che devono essere “spiegate”, per questo il processo di astrazione prevede il decidere quali caratteristiche sono rilevanti e quale formalismo descrittivo andrebbe adottato

In programmazione ci riferiamo alla descrizione astratta fornita da un modello (parte più alta) come alla **specifica** e al livello più basso nella gerarchia di modelli come **realizzazione**.

Le astrazioni utilizzate per il software si concentrano su cosa viene calcolato piuttosto che su come viene condotto il procedimento di calcolo.
Il passo finale è la verifica del funzionamento corretto tramite costrutti.

Astrarre per decomporre, rappresentare, generalizzare

Il processo di astrazione avviene sempre quando si **generalizza un concetto o un metodo**. Rendere generale un metodo solutivo messo a punto per un problema particolare **impone di comprendere l'essenza del metodo stesso**, prescindendo dalle peculiarità del caso specifico, in questo modo si tende ad individuare un modello unico che si adatti a situazioni diverse, magari attraverso l'impostazione corretta di parametri che permettano di utilizzare la soluzione generale al caso particolare.

Astrazione sui dati e sulle funzioni

Tali astrazioni portano immediatamente a due tipi di astrazioni fondamentali:

- **L'astrazione sui dati**, che consente di far riferimento a strutture algebrico-matematiche, caratterizzate da valori e da operazioni su tali valori, prescindendo quindi i costrutti specifici di un linguaggio e concentrandosi solo sulle proprietà **logiche** e **matematiche** delle strutture dati
- **L'astrazione sulle funzioni**, che consente di concentrare l'attenzione su **cosa** fa una particolare operazione piuttosto che sul **come** è fatta.
  Esempio: il richiamo di un operatore (o funzione), quando questa si richiama è già astratta di suo (non interessandoci di come funziona)

Tecniche di Progettazione

Le tecniche di progettazione si suddividono in diverse categorie, ciascuna con obiettivi specifici nel processo di sviluppo del software:

- Le **tecniche di specifica** consentono di esprimere gli elementi essenziali dello schema concettuale mediante formalismi grafici oppure attraverso un linguaggio basato sulla logica e sull'algebra per descrivere gli aspetti concettuali dei tipi astratti di dato.
- Le **tecniche di programmazione** riguardano i metodi per la strutturazione e per la stesura dei programmi (come la programmazione ad oggetti, gli schemi iterativi e ricorsivi etc.)
- La **modularizzazione** consente di razionalizzare lo sviluppo del software costruendo programmi costituiti da parti indipendenti e interagenti, in modo da facilitare la

manutenzione e l'evoluzione nel tempo
- Le **tecniche di progettazione di algoritmi e strutture dati** riguardano la definizione degli algoritmi e l'individuazione delle strutture dati più appropriate per un determinato problema
- Esistono tecniche di progettazione per specifiche classi di applicazioni, che sono specificatamente definite per applicazioni particolari.

Classificazione dei Linguaggi di Programmazione

I linguaggi di programmazione possono essere classificati secondo diversi paradigmi, ciascuno dei quali riflette una diversa filosofia di approccio alla risoluzione dei problemi:

- I **linguaggi imperativi** sono caratterizzati dal fatto che un programma corrisponde alla specifica di un insieme di istruzioni che corrispondono a precisi comandi impartiti ad una macchina che li esegue ripetitivamente.
- Nei **linguaggi di programmazione funzionale** un programma corrisponde alla specifica di una funzione che, in base a un insieme di dati in ingresso, calcola il risultato secondo una legge specificabile in modo matematico. Questo paradigma si basa sul concetto matematico di funzione e sull'applicazione di funzioni.
- I **linguaggi di programmazione logica** vedono un programma come la specifica di una relazione che sussiste tra un insieme di dati, e la specifica è costruita mediante un sistema formale basato sulla logica matematica.
- Nei **linguaggi di programmazione orientata agli oggetti**, un programma corrisponde alla specifica di un insieme di oggetti che rappresentano gli elementi della situazione in gioco in un certo problema.
  Ciascun oggetto è specificato in termini di una struttura interna e di un insieme di operazioni tramite le quali si ottiene il comportamento voluto per risolvere il problema.

Modularizzazione

(Def. in Tecniche di Progettazione)

Tra i diversi tipi di modularizzazione, grande importanza ha la modularizzazione per **tipo astratto**, in cui vengono sviluppati moduli che realizzano tipi astratti di dati significativi per l'applicazione specifica (come una banca o una biblioteca)

Un **modulo** è un’unità di programma con una struttura interna definita per uno scopo e che offre all'esterno un insieme prefissato di servizi utilizzabili da altri moduli;
un modulo è caratterizzato dalla sua struttura interna (insieme di tipi, variabili e funzioni), dall'insieme di servizi che esporta, dalle modalità con cui tali servizi possono essere utilizzati (l’interfaccia) e dall'insieme di servizi che importa da altri moduli per svolgere le proprie funzioni.

La qualità della modularizzazione migliora:

- All'aumentare della **coesione** del modulo, cioè quando il modulo incapsula caratteristiche omogenee sufficientemente indipendenti dagli altri moduli
- Con l’uso dell’**information hiding**, per cui i dettagli interni del modulo non devono essere rilevanti per chi lo usa
- Quando **l’accoppiamento tra moduli è basso**, evitando dipendenze non necessarie come l’uso diffuso di variabili globali
- Con **l’interfacciamento esplicito** che suggerisce di rappresentare tutti i dati scambiati tra sottoprogrammi tramite parametri.