

Consegne di MTDI

di Giordano Davide

Sommario

Consegna 1	1
Consegna 2	7
Fase 1: Distribuzione degli Pseudo-Algoritmi e lavoro individuale	7
Fase 2: Discussione collettiva e sintesi della definizione	8
Fase 3: Approccio alla definizione formale	9
Obiettivi di Apprendimento Operativi	10
Temi e Contenuti Informatici Affrontati	10
Consegna 3	11
Consegna 4	17
Analisi del codice e diagramma	18
Domande	19
Consegna 5	26

Consegna 1

Partendo dagli articoli elencati sotto e dagli argomenti discussi nelle prime lezioni, elencate:

- 3 punti/spunti/idee sulle quali siete d'accordo e perché' (riportate la frase e l'articolo da cui proviene)
 - È cambiata la natura degli artefatti, delle macchine, che produciamo. Non sono più artefatti fisici, sono “*artefatti cognitivi dinamici*”, azione congelata che viene sbloccata dalla sua esecuzione in un computer e genera conoscenza come risultato di tale esecuzione. La conoscenza statica dei libri diventa conoscenza dinamica nei programmi. Conoscenza in grado di produrre automaticamente, senza l'intervento umano, nuova conoscenza.
 - Ma la realizzazione dei computer, grazie all'informatica, consente di avere, per la prima volta nella storia dell'umanità, un sistema automatico

che, manipolando simboli di cui ignora il significato secondo istruzioni di cui ignora il significato, trasforma dati che hanno significato per l'uomo.

- In questi paesi si considera, correttamente, l'attuale società digitale come un'evoluzione della società industriale. Quest'ultima è stato il risultato della trasformazione della società agricola sotto la spinta della rivoluzione industriale. La società digitale è ancora una società di macchine, ma di tipo digitale, le "macchine cognitive".
- almeno un punto (meglio ancora 2 o 3) sui quali non siete d'accordo
 - Devono essere stabilite norme, regole e leggi efficaci, basate sul dibattito pubblico e su un ampio consenso. Queste norme, regole e leggi devono garantire accuratezza predittiva, equità e uguaglianza, responsabilità e trasparenza dei programmi software e degli algoritmi.
 - Le università sono il luogo in cui si producono nuove conoscenze e si coltiva il pensiero critico. Hanno quindi una responsabilità speciale di cui devono essere consapevoli.

Cerchiamo di scendere maggiormente nel dettaglio, spiegando cosa mi ha stupito e cosa invece meno, a partire proprio dalle cose positive a mio avviso.

Ritengo il primo punto (tratto da "La terza rivoluzione del potere") davvero formidabile in primis per un fattore semplicissimo: in un contesto storico in cui l'astrazione tende ad andare per la maggiore, l'autore sceglie di fare un po' di chiarezza prediligendo il materialismo. I computer (come ogni strumento digitale) non sono altro che artefatti, che, citando la Treccani, sono "Opere che derivano da un processo trasformativo intenzionale da parte dell'uomo". Nulla di più di prodotti dell'intelligenza umana, non diversi dalla prima ruota o dalle forchette per quanto riguarda la pura composizione materialistica (si passi ovviamente la metafora), nulla di più di una creazione particolarmente brillante.

In seconda reputo davvero interessante anche la seconda parte della citazione a Nardelli: cosa distingue i calcolatori da ogni macchina o artefatto per riprendere mai creato dall'uomo sta nella capacità di tali creazioni di generare conoscenza. Un prodotto ben diverso da quello che crea un trattore per la semina del grano, ben distante dal prodotto di un telaio, peraltro entrambi macchinari che hanno cambiato la storia dell'umanità con il loro apporto tecnologico. Consentono di creare prodotti, come dice lui, dinamici, che è un aspetto davvero trascurato e che io stesso davo per scontato prima di leggere l'articolo: basti prendere l'esempio di YouTube, con le miriadi

di milioni di minuti con corsi di qualsiasi campo si voglia approfondire, dalla programmazione alla biologia, fino a fisica e tantissimi altri campi in cui la divulgazione scientifica sta uscendo allo scoperto.

Continuando a leggere, viene esplicitato un altro grande concetto, affatto banale e che credo possa di nuovo essere indicativo della potenza e della grandezza di cosa abbiamo fra le mani: i computer non parlano la nostra lingua, fattore che a mio avviso è incredibile e un po' spaventoso. Penso banalmente al fatto che se mi presentassero un prodotto con sembianze non-umane e che non parla la mia lingua griderei all'invasione marziana.

Al di là delle assurdità, prima di iniziare a studiare informatica io non avevo minimamente idea di ciò, e forse è per questo che trovo tale spunto particolarmente interessante: sono troppo poche le nozioni date agli studenti che non provengono da istituti tecnici a riguardo del mondo informatico (e non digitale). Come discusso in aula e in accordo con quanto scritto nel manifesto di Vienna, reputo infatti che ci sia grande necessità di lavorare sull'approccio della scuola (e in generale del mondo della didattica) a tutto il complesso e affascinante mondo dell'informatica, anche solo per conoscenza generale del mondo che circonda i futuri studenti. D'altronde, studiamo fin da piccoli lingue straniere che possibilmente fruiremo meno di quanto invece non faremo mai con un computer, perché allora non avvicinarsi alla lingua di questi strumenti fondamentali?

Per concludere, trovo decisamente interessante questa visione appartenente a paesi quali Stati Uniti e Regno Unito di una moderna società come evoluzione di quanto visto nella società industriale. Reputo in un certo senso evidente che l'approccio da loro adottato sia il più efficiente, ma lo penso semplicemente in relazione a quanto visto dalla storia: è stato un metodo vincente nella prima e nella seconda rivoluzione tecnologica, ma non solo, in quanto persino oggi si vede che un atteggiamento particolarmente propositivo nei confronti del nuovo settore ha portato i maggiori vantaggi nei confronti del resto del mondo: si veda la tecnologia americana e quello che ormai è in grado di fare (tutto merito dello slancio ottenuto negli anni '50 nella ricerca e sperimentazione). Ogni società a mio avviso rappresenta la naturale evoluzione di cosa diviene nel presente e guardando l'inizio della rivoluzione tecnologica e gli strumenti utilizzati non mi pare un paragone azzardato quello scritto da Nardelli: basti pensare che le schede perforate utilizzate nei telai hanno rappresentato la chiave di Volta nella costruzione dei computer moderni!

Passando invece a cosa non mi ha particolarmente convinto, le questioni da tirare in ballo sono due:

- Per quanto riguarda lo stabilire norme e regole efficaci, non sono assolutamente contrario all'idea, anzi tutt'altro. Cosa mi fa dubitare è la realizzabilità di questo proposito: è vero che gli strumenti a nostra disposizione consentono di comunicare attraverso reti in pochi millisecondi fino al capo più remoto della terra, ma è anche vero che accordare quasi 9 miliardi di menti possa essere particolarmente arduo;
 - A riguardo della questione università e delle responsabilità che le si vorrebbe attribuire, io credo che piuttosto che addossare un carico così elevato a un sistema di istruzione che NON è accessibile a tutti (si guardino anche solo i costi di mantenimento di rette, vitto e alloggi per uno studente) sia un errore, anche grossolano. Reputo invece che diluire le responsabilità su tutti i livelli di istruzione (dalla primaria all'ultimo ciclo di istruzione) possa essere più utile;
-

Qui sotto trovate una lista di criteri estratti dall'articolo [The Science in Computer Science](#). Sono usati dall'autore per definire la credibilità di un settore come "scienza".

- Organized to understand, exploit, and cope with a pervasive phenomenon.
 - Encompasses natural and artificial processes of the phenomenon.
 - Codified structured body of knowledge.
 - Commitment to experimental methods for discovery and validation.
 - Reproducibility of results.
 - Falsifiability of hypotheses and models.
Diritti d'autore e intelligenza artificiale
 - Ability to make reliable predictions, some of which are surprising.
Unione di statistica e informatica
- L'informatica soddisfa qualcuno dei criteri sopra elencati? Se sì quali? e perché?

L'informatica, in quanto disciplina, soddisfa diversi dei criteri elencati nell'articolo *The Science in Computer Science* a mio avviso. Vediamo quali e perché:

1. Organized to understand, exploit, and cope with a pervasive phenomenon:

A mio avviso l'informatica è indubbiamente legata alla definizione stessa di fenomeno pervasivo: per definizione questo termine indica qualcosa che tende a dominare, ma cosa meglio del concetto di informazione può rappresentare questa definizione? D'altro canto, l'informatica altro non è (citando la Treccani) che " La scienza che si occupa dell'ordinamento, del trattamento e della

trasmissione delle informazioni...” Dunque, un campo della scienza che lavora su un elemento pervasivo non può che esserlo per proprietà transitiva, in quanto consente la sua trasmissione, il suo apprendimento e il suo tramandarsi.

2. Encompasses natural and artificial processes of the phenomenon:

Trovo esemplificativa la storia dietro il riconoscimento in quanto scienza dell’informatica per spiegare di come essa stessa possa essere utile al fine di comprendere meglio i processi naturali: se originariamente la scienza in toto si è schierata contro, con lo sviluppo delle tecnologie informatiche e il passare degli anni sempre più macro-aree scientifiche si sono aperte alle innovazioni portate dal ramo informatico, come David Baltimore, premio Nobel per la biologia nel 1975, che definiva la traduzione del DNA come un processo informativo naturale. Ad oggi, i più grandi passi avanti nella medicina di precisione sono stati resi possibili dall’utilizzo massivo di strumenti informatici e intelligenze artificiali. Uscendo dal ramo della biologia, anche per quanto riguarda il settore della geologia, la precisione nelle previsioni di eventuali fenomeni sismici è visibilmente aumentata, con conseguente diminuzione del tasso di mortalità a seguito di questo tipo di sventure;

3. Codified structured body of knowledge:

Nella sua stessa struttura l’informatica risulta essere una struttura organizzata, ovviamente in maniera diversa a seconda di cosa ci troviamo davanti: solo affacciandoci al mondo informatico abbiamo la gerarchia della memoria, che risulta uno dei più semplici modelli a piramide, ma anche organizzazioni decisamente più complesse, come quelle che vengono proposte nelle reti Client-Server, che rappresentano, come dice il Professor Ciravegna nel suo corso di Tecnologie Web, “un sistema sociale più che umano”, in quanto può venir facilmente rappresentato dal sistema di gestione degli ordini di una qualunque catena di fast food odierna. Se vogliamo poi addentrarci ancora di più, il mondo stesso della codifica in qualsiasi sistema rappresenta un sistema estremamente ordinato e funzionale, il quale permette di creare e rappresentare pressoché qualsiasi elemento della nostra vita quotidiana.

4. Reproducibility of results:

Trovo abbastanza ridicolo in un certo senso parlare della possibilità di riprodurre facilmente il lavoro fatto utilizzando un qualsiasi strumento informatico, ma per il semplice fatto che il solo possedere uno di questi dispositivi consente di riprodurre in qualsiasi momento e in qualsiasi posizione ci si trovi ogni contenuto esistente in rete, tra l’altro anche con la possibilità di farlo contemporaneamente

a più persone e su più dispositivi. Del resto, se programmando una delle prime cose insegnate è l'iterazione per eseguire una stessa operazione su una o più strutture dati, un motivo ci sarà.

5. **Falsifiability of hypotheses and models:**

L'informatica più di altre discipline e campi soddisfa bene il concetto di falsificabilità, ovvero l'idea che una teoria debba poter essere falsa e introdotta dal filosofo Karl Popper. Prendiamo l'intelligenza artificiale: quando si sviluppa un modello di machine learning per riconoscere le immagini, ad esempio, si crea un sistema che fa delle previsioni basate sui dati. Se il modello prevede male, ad esempio scambia un gatto per un cane in molte foto, sappiamo che c'è qualcosa che non va e il modello può essere "falsificato". A quel punto, si torna indietro al modello per correggerlo o cambiarlo. Un esempio più interessante a cui pensavo è legato al copyright. Supponiamo che io crei un'IA che produce opere d'arte o brani musicali. Se qualcuno sostiene che l'IA ha violato il copyright, posso testare questa affermazione confrontando ciò che l'IA ha creato con altre opere d'arte preesistenti. Se risulta che l'IA abbia copiato esattamente una parte dell'opera d'arte protetta da copyright, l'idea che operi con le regole del copyright è falsificata. In definitiva, il computer testa continuamente le teorie, e se queste sono false o violano le regole, come quelle del copyright, le rilevano e le correggono facilmente.

6. **Ability to make reliable predictions, some of which are surprising:**

A mio avviso, l'informatica soddisfa il criterio della capacità di fare previsioni affidabili: ne sono un esempio i modelli predittivi, che permettono di creare previsioni accurate in campi come il machine learning e l'intelligenza artificiale. Inoltre, negli ultimi anni moltissimi algoritmi di apprendimento supervisionato sono stati addestrati su dati storici per prevedere comportamenti futuri come il riconoscimento di immagini o le preferenze degli utenti. Ad aggiungersi a ciò, le simulazioni consentono di modellare sistemi complessi come il clima o i sistemi economici per produrre previsioni affidabili in condizioni diverse. Alcune previsioni risultano sorprendenti per via della scoperta di pattern nascosti grazie a tecniche avanzate che rivelano correlazioni o tendenze inaspettate. Gli algoritmi di deep learning come le reti neurali talvolta generano previsioni che sorprendono gli esperti individuando caratteristiche nei dati che non erano evidenti o prevedibili.

Elenco articoli:

- [Informatica: la terza rivoluzione "dei rapporti di potere"](#)
- [Informatica e competenze digitali: cosa insegnare?](#)
- [MANIFESTO DI VIENNA PER L'UMANESIMO DIGITALE](#)

Consegna 2

Rivedere l'attività sugli [pseudoalgoritmi](#) in fasi indicando:

- [proposta di suddivisione in fasi](#) esplicitando in particolare per ciascuna fase **Consegna e conclusione**
 - per quanto riguarda le **modalità di Svolgimento e discussione** scegliere una delle modalità proposte in [questo file](#) motivando la scelta
 - provare a delineare degli obiettivi di apprendimento operativi che possono essere raggiunti al termine di questa attività
 - identificare i temi, contenuti informatici affrontati in questa attività
-

La strategia che a mio avviso sembra meglio conformarsi allo schema delineato per l'attività sugli pseudo-algoritmi è il **Problem-Based Learning (PBL)**: stimolando il pensiero critico e l'analisi individuale e collettiva, caratteristiche chiave nel processo di comprensione e definizione del concetto di algoritmo, permette agli studenti di esplorare un problema complesso e senza una soluzione univoca, che nel nostro caso sarebbe determinare se uno pseudo-algoritmo possa essere considerato un vero algoritmo. Vediamo come nell'analisi:

Fase 1: Distribuzione degli Pseudo-Algoritmi e lavoro individuale

Nella prima fase del lavoro abbiamo:

- **Consegna:** Ogni studente, lavorando individualmente o in coppie, riceve un foglio contenente una lista di pseudo-algoritmi. Questi, che per la maggior parte sono sequenze di istruzioni generiche (es. “Ricetta del ragù”), sono concepiti per assomigliare a veri algoritmi, ma non soddisfano necessariamente tutti i criteri essenziali richiesti per essere considerate tali.
- **Svolgimento:** Il compito degli studenti è esaminare attentamente ciascuno degli pseudo-algoritmi forniti, discutendo e confrontando le proprie idee con il compagno di lavoro, al fine di stabilire se ogni sequenza possa effettivamente essere definita come un algoritmo. Gli studenti saranno spinti a cercare di individuare in maniera autonoma una definizione di algoritmo in base al rispetto

o meno di determinate caratteristiche fondamentali, come la finitezza, ossia la capacità dell'algoritmo di produrre un risultato in un tempo limitato, la non ambiguità, e l'effettività. Questo lavoro richiede loro di esplorare il concetto di algoritmo in modo critico, senza ricevere definizioni o risposte preconfezionate – secondo l'utilizzo della strategia PBL.

- **Discussione:** La discussione in questa fase si limiterà ad essere soltanto tra studenti, ma sarebbe consigliabile per l'insegnante muoversi tra i banchi, ascoltando con attenzione le conversazioni tra studenti e intervenendo solo quando necessario per fornire suggerimenti o incoraggiamenti, al fine di indirizzare gli studenti ad evidenziare determinati aspetti che potrebbero fornire spunti interessanti nella fase successiva del lavoro. Infatti, l'obiettivo è che ogni analisi aggiunga un tassello alla costruzione di una definizione condivisa e completa di algoritmo. Ecco un esempio di interazione tipo: l'insegnante potrebbe presentare uno pseudo-algoritmo volutamente ambiguo e chiedere: "Le istruzioni qui sono abbastanza chiare da non creare confusione? In caso contrario, come potrebbero essere migliorate per essere più precise?" Un'altra domanda, rivolta a un algoritmo che potrebbe andare avanti indefinitamente, potrebbe essere: "Se seguiamo queste istruzioni, arriveremo sempre a un risultato, oppure c'è la possibilità che il processo non finisca mai?" In questo modo, gli studenti sono invitati a interrogarsi in maniera critica su ciascun aspetto e a scoprire gradualmente i criteri distintivi di un vero algoritmo.
- **Conclusione:** Il docente sintetizza i punti chiave emersi, riassumendo i concetti principali (finitezza, non ambiguità, effettività) e assicurandosi che gli studenti abbiano compreso il compito.

Fase 2: Discussione collettiva e sintesi della definizione

- **Consegna:** L'insegnante, durante l'attività, guida progressivamente gli studenti verso una definizione precisa e condivisa di algoritmo, utilizzando i commenti raccolti durante lo svolgimento e arricchendo il dialogo con domande di carattere generale (seguendo quanto esplicitato secondo la strategia PBL): ogni pseudo-algoritmo proposto è infatti scelto appositamente per illustrare una caratteristica chiave degli algoritmi – come la finitezza, la chiarezza, o la concretezza del risultato – e ogni domanda si concentra su questo aspetto, stimolando gli studenti a riconoscere le differenze e le particolarità di ciascun caso.
- **Svolgimento:** Il docente deve adottare un approccio maieutico alla consegna: deve a mio avviso guidare gli studenti verso la costruzione di una comprensione autonoma e profonda del concetto di pseudo-algoritmo. Invece di fornire spiegazioni dirette, l'insegnante deve favorire il ragionamento critico e consentire

ad ogni studente di tentare di arrivare da solo a una definizione più o meno precisa attraverso una serie di quesiti progressivi, ciascuno formulato per evidenziare un aspetto cruciale che caratterizza un algoritmo rispetto a una sequenza di istruzioni generiche.

- **Discussione:** Infine, l'insegnante riunisce gli elementi emersi da queste riflessioni per condurre la classe verso una definizione quanto più precisa e completa di algoritmo, capace di comprendere tutte le caratteristiche fondamentali emerse durante l'attività: finitezza, non ambiguità, effettività e capacità di risolvere un problema in modo efficace. Aggiungendo ogni tassello con domande mirate, l'insegnante assicura che ogni studente contribuisca alla costruzione della definizione e abbia ben chiara la natura complessa di un vero algoritmo. Questo approccio permette agli studenti di acquisire una conoscenza approfondita e articolata del concetto, sviluppando anche competenze di analisi critica e di riflessione indipendente che saranno preziose per affrontare successivamente argomenti più complessi.
 - **Conclusione:** Alla fine dell'attività, ogni studente avrà sviluppato una comprensione più profonda e sfumata del concetto di algoritmo, avendo costruito questa conoscenza in modo attivo e consapevole attraverso il dialogo e la riflessione condivisa.
-

Fase 3: Approccio alla definizione formale

- **Consegna:** l'insegnante propone una definizione generale e universale di "algoritmo," tratta da un dizionario o da una fonte accademica, allo scopo di consolidare i concetti appresi e verificare la completezza della comprensione da parte degli studenti. La lettura della definizione formale offre un punto di riferimento chiaro e aiuta a confrontare le caratteristiche di un algoritmo precedentemente emerse durante l'attività con quelle incluse nella definizione universale. Questo confronto favorisce una riflessione sui punti chiave discussi, come la finitezza, la chiarezza e univocità delle istruzioni, e l'effettività.
- **Svolgimento:** Per applicare concretamente la definizione e verificare la comprensione, l'insegnante introduce un piccolo algoritmo reale e funzionante, scelto con cura per essere pertinente al ramo di studi della classe: ad esempio, in una classe di informatica, un algoritmo di ordinamento semplice, come il bubble-sort, o in un contesto più generico, si potrebbe proporre un algoritmo di quelli visti precedentemente ma corretto seguendo le linee guida definite durante la precedente analisi. Dunque, la scelta dell'algoritmo si orienta verso un'attività che sia in qualche modo familiare agli studenti, così da mantenere il discorso collegato a contesti di vita reale e alle loro esperienze

- **Discussione:** Il docente facilita una discussione di gruppo, guidata da slide o dal libro di testo, per esaminare in dettaglio l'algoritmo proposto e verificarne la corrispondenza con la definizione formale. Si esplorano le caratteristiche chiave dell'algoritmo, come la finitezza, la chiarezza delle istruzioni, e l'efficacia nella risoluzione di un problema. L'insegnante stimola il confronto tra gli studenti, chiedendo loro di individuare eventuali incongruenze o ambiguità e suggerire miglioramenti. Questo passaggio permette di consolidare le conoscenze e chiarire dubbi, in conformità con la strategia del PBL. Il tutto viene eseguito con l'ausilio di slide proiettate o con il libro di testo o in caso di necessità di materiale fornito dal docente stesso.
- **Conclusione:** La lezione si conclude con il chiarimento di eventuali dubbi e ribadendo i concetti fondamentali per fissarli ulteriormente.

Obiettivi di Apprendimento Operativi

Al termine dell'attività, gli studenti dovrebbero essere in grado di:

- **Distinguere tra un vero algoritmo e una sequenza di istruzioni generiche,** identificando le caratteristiche fondamentali (finitezza, non ambiguità, effettività).
- **Articolare una definizione precisa e motivata di "algoritmo",** utilizzando termini appropriati e basandosi su esempi concreti.
- **Analizzare e correggere sequenze di istruzioni non strutturate,** trasformandole in algoritmi attraverso la rimozione di ambiguità e la garanzia di finitezza.
- **Collaborare efficacemente con un compagno,** discutendo e confrontando opinioni per arrivare a una comprensione condivisa del concetto di algoritmo.
- **Applicare la definizione di algoritmo a contesti pratici,** riconoscendo l'importanza delle sue caratteristiche fondamentali nella risoluzione di problemi reali.

Temi e Contenuti Informatici Affrontati

1. **Definizione di Algoritmo:** Introduzione alle caratteristiche fondamentali (finitezza, non ambiguità, effettività).
2. **Analisi di Pseudo-Algoritmi:** Comprensione delle differenze tra sequenze generiche di istruzioni e algoritmi strutturati.
3. **Applicazione di Criteri di Valutazione:** Sviluppo di capacità di valutazione critica utilizzando criteri definiti per giudicare la qualità di un algoritmo.
4. **Introduzione a Esempi di Algoritmi Reali:** Studio di semplici algoritmi (es. algoritmi di ordinamento) per consolidare la teoria attraverso la pratica.

Consegna 3

Scegliere un [compito di programmazione](#) non troppo complesso (come esempio si veda il [rainfall problem](#)).

Scomporre il compito in sotto obiettivi, fino ad arrivare a individuare per ciascun sotto obiettivo delle soluzioni paradigmatiche che si basano su concetti di base della programmazione: iterazione, selezione, gestione dell'input e dell'output.

Consideriamo adesso le seguenti conoscenze e abilità:

- saper suddividere un problema in sottoproblemi per cui esistono delle soluzioni paradigmatiche di base
- conoscere una serie soluzioni paradigmatiche da utilizzare per la soluzione di problemi semplici e ricorrenti

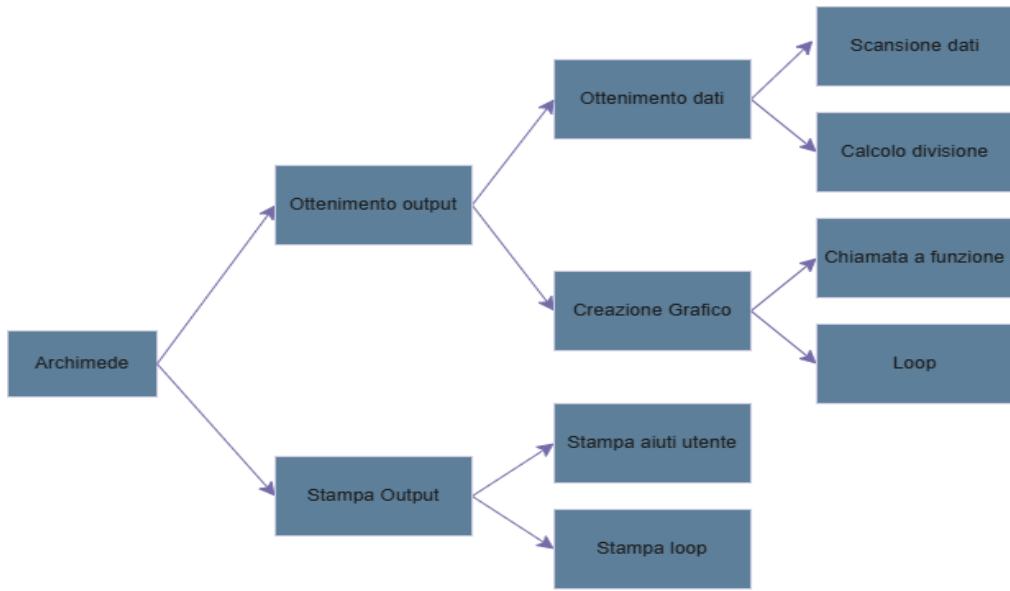
Il lavoro verrà svolto sul codice “C” che calcola la spirale di Archimede, che, citando Wikipedia, “...è una curva che parte da un punto centrale (il "polo") e si avvolge intorno a esso allontanandosi progressivamente. La distanza tra i bracci successivi della spirale aumenta in modo lineare, quindi ogni giro successivo è equidistante dal precedente. La sua equazione in coordinate polari è:

$$r = a + b \cdot \theta$$

dove:

- **r** è la distanza dal polo,
- **θ** è l'angolo (in radianti),
- **a** e **b** sono costanti che determinano la posizione e la spaziatura della spirale.”

Cerchiamo ora di lavorare per suddividere le varie sezioni del codice in sottoproblemi



Vediamo dunque insieme le varie fasi nel codice:

- Ottenimento output
 - Calcolo dati
 - Scansione dati
 - **Calcolo divisione**
 - Creazione Grafico
 - **Chiamata a funzione**
 - **Loop**
- Stampa Output
 - **Stampa per Utente**
 - **Loop**

```

#include<graphics.h>
#include<stdio.h>
#include<math.h>

#define pi M_PI

int main(){
    double a,b,cycles,incr,i;

    int steps,x=500,y=500;

    printf("Enter the parameters a and b : ");
    scanf("%lf%lf",&a,&b);

    printf("Enter cycles : ");
    scanf("%lf",&cycles);

    printf("Enter divisional steps : ");
    scanf("%d",&steps);

    incr = 1.0/steps;

    initwindow(1000,1000,"Archimedean Spiral");

    for(i=0;i<=cycles*pi;i+=incr){
        putpixel(x + (a + b*i)*cos(i),x + (a + b*i)*sin(i),15);
    }

    getch();

    closegraph();
}

```

In alcuni articoli di didattica dell'informatica si propone di far acquisire le conoscenze e le abilità sopra elencate (saper suddividere un problema in sottoproblemi per cui esistono delle soluzioni paradigmatiche di base, conoscere una serie soluzioni paradigmatiche da utilizzare per la soluzione di problemi semplici e ricorrenti) prima di iniziare a programmare in un linguaggio specifico in modo da acquisire capacità di problem solving senza doversi perdere nei dettagli e dei diversi livelli di astrazione di un singolo linguaggio. Cosa ne pensate?

Reputo importante che un problema prima di essere risolto venga scomposto in sottoproblemi e questo grazie all'esperienza finora accumulata negli anni di studio dei diversi linguaggi di programmazione.

A mano a mano che i concetti vengono spiegati, il cumulo di informazioni ricevute è diventato, ma tuttora diventa, sempre più grande con il passare delle lezioni: finché però il lavoro non si è tradotto in realizzazione di progetti non ho realmente colto l'utilità di questa procedura nel suo vero senso. Trovo utile raccontare l'esperienza avuta con il progetto di Sistemi operativi: il linguaggio da utilizzare era UNIX unito a C e, avendo seguito attentamente le lezioni, non ero realmente preoccupato di quello che poi sarebbe stato il primo lavoro di grosse dimensioni della mia vita. Ben presto però ho iniziato a rendermi conto che la effettiva difficoltà non fosse tanto il conoscere il linguaggio in sé, quanto piuttosto avere bene in mente il quadro della situazione: senza una attenta analisi di quali fossero i problemi e le modalità con cui ognuno dei file dovesse eseguire il suo compito, non sarebbe stato possibile fare nulla.

Sono d'accordo anche sull'apprendere la seconda abilità prima di iniziare a lavorare, ossia quella dello studio di determinate serie di soluzioni paradigmatiche per risolvere piccoli problemi ricorrenti: specialmente chi si approccia da novizio al mondo dell'informatica ha molto spesso poche idee e ben confuse nei primi tempi (e parlo nuovamente per esperienza personale). Nel mio percorso, un grandissimo apporto lo ha avuto partire da piccole porzioni di codice molto spesso ripetute in metodi e funzioni decisamente più grandi: in particolare nello studio di Java a Programmazione 1 la comprensione totale della manipolazione di array e matrici è passata attraverso l'utilizzo fisso di massimo 10 righe di codice, sempre uguali, ma estremamente duttili. Lo reputo davvero utile perché consente di illuminare la via molto spesso a quelli che poi saranno ragionamenti più complessi o addirittura l'approccio a un nuovo linguaggio.

Per quanto dunque spesso possa sembrare più semplice e rapido lanciarsi sui problemi con gli occhi bendati alla ricerca di una soluzione, dunque, trovo sia meglio la strategia della pianificazione attenta e meticolosa, possibilmente accompagnata da una altrettanto attenta scomposizione del lavoro da fare, perché in moltissimi casi il problema che ci stiamo ponendo non è un ostacolo tanto quanto il sottovalutare la complessità di un piccolo ciclo, anche uno di quelli potenzialmente banali per contare i numeri dispari.

Scegliere un esercizio/problema abbastanza complesso dalla pagina [Rosetta Code](#) o dagli esercizi posti per le [olimpiadi di informatica](#). Identificare i pattern algoritmici ed elementari coinvolti nella soluzione del problema.

Il testo, preso dall'elenco degli esercizi preparati per le olimpiadi di informatica, è questo:

“ [Parole periodiche \(periodicwords\)](#) ”

Una stringa **s** si dice periodica se esiste una stringa **t** tale che **sss** può essere ottenuta concatenando più copie (almeno 2) di **t**.

In altre parole, **s** è periodica se $s = t + t + \dots + t$ per qualche stringa $t \neq s$, dove $+$ rappresenta l'operazione di concatenazione tra stringhe.

Si considera una stringa $A = a_0 a_1 \dots a_{N-1}$ di lunghezza **N** e **Q** interrogazioni della forma **li**, **ri**. Per ogni interrogazione, determinare se la sottostringa $A[li \dots ri] = a_{li} a_{li+1} \dots a_{ri}$ è una stringa periodica.

Input

- La prima riga contiene un numero intero **N**.
- La seconda riga contiene una stringa **S** di lunghezza **N**.
- La terza riga contiene un numero intero **Q**.
- Le successive **Q** righe contengono i valori **li**, **ri** che descrivono le interrogazioni.

Output Per ogni interrogazione, stampare "YES" se la sottostringa richiesta è una stringa periodica o "NO" se non lo è. "

Di seguito una mia implementazione dell'esercizio, cui segue l'analisi dei vari pattern elementari/algoritmici.

```

27 // Ciclo per ogni interrogazione
28 for (int i = 0; i < Q; ++i) {
29     int start = l[i];
30     int end = r[i];
31
32     // Estrai la sottostringa
33     String substring = S.substring(start, end + 1);
34
35     // Verifica la periodicità della sottostringa
36     if (isPeriodic(substring)) {
37         ans[i] = "YES";
38     } else {
39         ans[i] = "NO";
40     }
41 }
42
43 // Stampa i risultati
44 for (int i = 0; i < Q; ++i)
45     prnt.format(format:"%s\n", ans[i]);
46 prnt.format(format:"\n");
47
48 fout.flush();
49
50
51 // Funzione per verificare la periodicità di una sottostringa
52 private static boolean isPeriodic(String substring) {
53     int len = substring.length();
54
55     // Prova tutte le possibili lunghezze di periodo fino a metà della lunghezza della sottostringa
56     for (int period = 1; period <= len / 2; period++) {
57
58         if (len % period == 0) { // Se period divide la lunghezza della sottostringa
59             String pattern = substring.substring(beginIndex:0, period); // estraie da 0 a period
60             StringBuilder repeatedPattern = new StringBuilder(); // vuoto
61
62             // Costruisci la stringa ripetendo il pattern
63             for (int j = 0; j < len / period; j++) {
64                 repeatedPattern.append(pattern);
65                 System.out.println(repeatedPattern);
66             }
67
68             // Se la sottostringa coincide con il pattern ripetuto, è periodica
69             if (repeatedPattern.toString().equals(substring)) {
70                 return true;
71             }
72         }
73
74     }
75
76     return false;
77 }
78

```

Per analizzare al meglio credo sia necessario fare distinzione in pattern trovati nel main (riporto solo il codice scritto da me e non la base di partenza fornita dal sito) e pattern del metodo “isPeriodic”, ausiliare per aiutarmi nella risoluzione. Dunque:

- **Main:**
 - **Primo loop:** Process Items Until Done, in quanto cicla prendendo come limite il numero di elementi fornito dall’utente (Q), prepara con il for la venuta dell’elemento successivo, la condizione da rispettare per l’esecuzione e l’eventuale chiusura del ciclo;

- **Selezione con metodo:** Alternative action, data la presenza stessa della condizione if/else (rispetta dunque a pieno il modello proposto);
- **Secondo loop:** Process All Items, in quanto si limita a effettuare un'unica operazione uguale per tutti gli elementi (rispettando il modello proposto);

- **isPeriodic:**
 - **Primo Loop:** Process Until Done, in quanto eseguo operazioni basandosi come limite sulla lunghezza della sottostringa, performando controlli per effettuare eventuali azioni;
 - **Prima Selezione:** Guarded Actions, in quanto ci si limita a verificare che la variabile-sentinella sia rispettata;
 - **Secondo loop:** Process All Items, in quanto si limita a effettuare un'unica operazione uguale per tutti gli elementi (rispettando il modello proposto);
 - **Seconda Selezione:** Process One Item, in quanto si limita a verificare l'uguaglianza della sottostringa calcolata con quella corrente da analizzare, dunque processa il dato e da in output un risultato.

Consegna 4

Scegliete un esercizio di programmazione (es. il [rainfall problem](#)).

Effettuare un'analisi (da un punto di vista didattico) dei costrutti usati nell'esercizio analogamente a quanto svolto nelle sezioni 7.2 e 7.3 dell'articolo [Developing Assessments to Determine Mastery of Programming Fundamentals](#).

Integrare l'analisi valutando le misconceptions che possono emergere sui concetti svolti. Trovate la lista delle misconceptions nell'appendice di questa tesi ([Tesi: Visual Program Simulation in Introductory Programming Education](#))

Il lavoro è stato svolto in gruppo, con Andriani Agnese. Il lavoro è finalizzato ad approfondire l'insieme di processi e tecniche utilizzati per esaminare, progettare, e valutare le attività di insegnamento e apprendimento. Lo scopo principale dell'analisi didattica è migliorare l'efficacia dell'insegnamento, identificare difficoltà di apprendimento e sviluppare strategie che facilitino la comprensione e l'acquisizione delle conoscenze da parte degli studenti.

Analisi del codice e diagramma

```
#include <stdio.h>
#include <ctype.h>

int can_make_words(char **b, char *word)
{
    int i, ret = 0, c = toupper(*word);

#define SWAP(a, b) if (a != b) { char *tmp = a; a = b; b = tmp; }

    if (!c) return 1;
    if (!b[0]) return 0;

    for (i = 0; b[i] && !ret; i++) {
        if (b[i][0] != c && b[i][1] != c) continue;
        SWAP(b[i], b[0]);
        ret = can_make_words(b + 1, word + 1);
        SWAP(b[i], b[0]);
    }

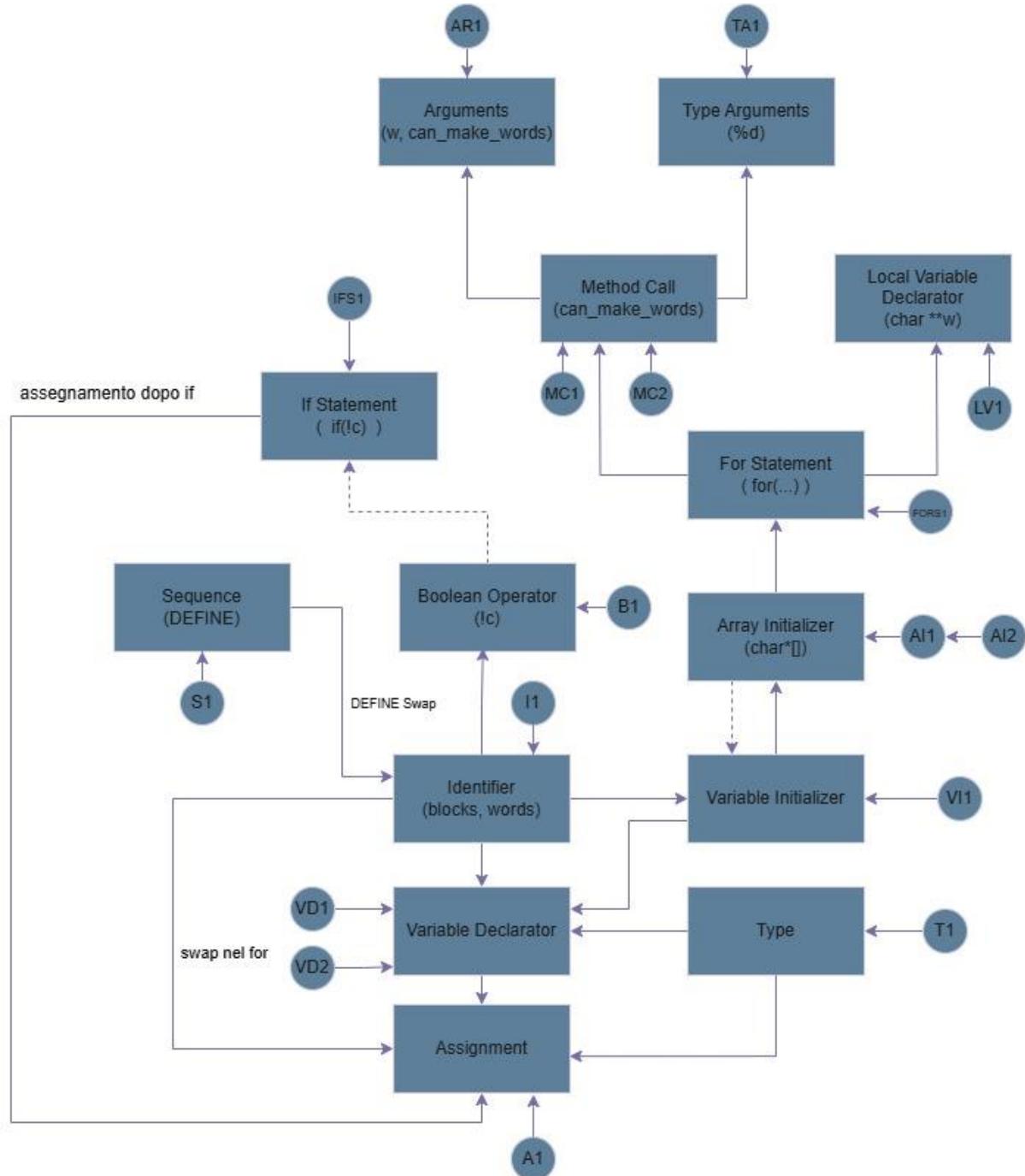
    return ret;
}

int main(void)
{
    char* blocks[] = {
        "BO", "XK", "DQ", "CP", "NA",
        "GT", "RE", "TG", "QD", "FS",
        "JW", "HU", "VI", "AN", "OB",
        "ER", "FS", "LY", "PC", "ZM",
        0 };

    char *words[] = {
        "", "A", "BARK", "BOOK", "TREAT", "COMMON", "SQUAD", "Confuse", 0
    };

    char **w;
    for (w = words; *w; w++)
        printf("%s\t%d\n", *w, can_make_words(blocks, *w));

    return 0;
}
```



Domande

Di seguito, ecco l'analisi didattica e la sua struttura a nostro avviso.

I1: E' importante che ogni identificatore abbia un nome diverso?

- Si, altrimenti il programma non potrebbe distinguere variabili e funzioni;
- Si, perché è più facile per un programmatore distinguerli;
- No, possono essere uguali purché si trovino in contesti differenti;
- No, gli identificatori non sono importanti se il codice funziona

Misconceptions:

- Non comprendere che identificatori duplicati in ambiti differenti possono coesistere ma necessitano di contesti distinti.
- Credere che gli identificatori non influenzino la funzionalità del programma, ignorando l'importanza della leggibilità del codice.
- Pensare che identificatori duplicati causino sempre errori di compilazione.

AI1: Quanti elementi contiene inizialmente l'array “blocks”?

Risposta: Contiene 21 elementi, 20 stringhe e 1 terminatore 0.

Misconceptions:

- Confondere l'array con una stringa terminata da \0.
- Presumere che gli array contengano sempre un "terminatore" esplicito.

AI2: Perché viene usato un carattere terminatore alla fine dell'array?

Risposta: Viene usato come controllo nei cicli per comprendere facilmente quando un array termina.

Misconceptions:

- Credere che il terminatore sia necessario per tutti i tipi di array. Solo array di caratteri che rappresentano stringhe richiedono un terminatore.

VD1: Che cosa succede se provi a utilizzare una variabile non inizializzata in C?

- Il programma compila e usa un valore di default.
- Il programma non compila.
- Il programma compila ma genera un errore a Runtime.

Misconceptions:

- Pensare che vengano assegnati automaticamente valori di default a tutte le variabili, incluse quelle locali. In realtà, le variabili locali non inizializzate contengono valori casuali derivati dal contenuto della memoria. Solo le variabili globali e statiche sono inizializzate automaticamente a zero.

VD2: È possibile dichiarare una variabile senza assegnarle immediatamente un valore?

- Sì, ma deve essere inizializzata prima di essere utilizzata.
- No, deve essere inizializzata al momento della dichiarazione.
- Sì, e può essere utilizzata anche senza inizializzazione.

Misconceptions:

- Presumere che l'inizializzazione sia obbligatoria al momento della dichiarazione per tutte le variabili.
- Non comprendere che la dichiarazione senza inizializzazione è permessa ma deve essere gestita con attenzione.

VI1: La porzione di codice: “int x = 1” è equivalente a “Int x; x = 1”?

Risposta: Si

Misconceptions:

- Credere che la dichiarazione con inizializzazione (int x = 1;) sia più efficiente. In realtà non c'è differenza in termini di esecuzione; la distinzione è puramente sintattica.
- Credere che “int x; x = 1;” non sia valido per variabili locali. Entrambe le sintassi sono valide, indipendentemente dal contesto (locale o globale).

T1: Ogni valore in C ha un tipo?

- Sì, sempre
- No, non tutti i valori hanno un tipo
- Solo i numeri hanno un tipo
- Dipende dal compilatore

Misconceptions:

- Credere che alcuni valori possano non avere un tipo.
- Pensare che il tipo sia definito solo al momento dell'uso e non al momento della dichiarazione.
- Pensare che alcuni valori, come i letterali numerici o NULL, non abbiano tipo.

A1: Quale delle seguenti assegnazioni sono valide per modificare il primo elemento di “blocks”?

- blocks[0] = “AB”;
- blocks[0] == “AB”;
- blocks = “AB”;
- “AB” = blocks[0]

Misconceptions:

- Confondere l'operatore di assegnamento (=) con l'operatore di confronto (==).

- Pensare che un elemento specifico di un array non possa essere modificato direttamente.

A2: Quale delle seguenti dichiarazioni è equivalente a $x = x + 10$?

- $x += 10$;
- $x =+ 10$;
- $x = 10 +$;
- Nessuna delle precedenti.

Misconceptions:

- Confondere $x += 10$ (operatore di assegnamento combinato) con $x =+ 10$, che assegna il valore positivo di 10 a x.
- Non comprendere l'importanza degli operatori di assegnamento combinato.

IFS1: Qual è il valore di x dopo l'esecuzione del seguente codice?

```
int x = 10;

if (x < 5) {
    x = x + 2;
} else {
    x = x - 3;
}
```

Risposte:

- 7
- 12
- 10
- 5

Misconceptions:

- Credere che l'istruzione $x < 5$ venga interpretata come "minore o uguale a 5".
- Pensare che il ramo else venga eseguito solo se if non compila.

FORS1: Qual è il valore di x dopo l'esecuzione del seguente codice?

```
int x = 0;  
for (int i = 0; i < 5; i++) {  
    x = x + i;  
}
```

Risposte:

- 10
- 5
- 15
- 0

Misconceptions:

- Non comprendere la logica di iterazione e accumulo delle variabili.
- Dimenticare che il primo valore di i è 0 e quindi il contributo iniziale a x è nullo.

B1: Qual è il valore di res alla fine dell'esecuzione?

```
bool first = true;  
bool second = false;  
bool res = first && second;
```

Risposte:

- true
- false
- null
- 1

Misconceptions:

- Confondere l'operatore && con ||
- Pensare che il risultato di un'operazione booleana possa essere null.

S1: Qual è il valore di x dopo l'esecuzione del seguente codice?

```
int x = 7;  
int y = 3;  
  
x = x - y;  
y = y * 2;  
x = x + y;
```

Risposte:

- x = 8, y = 6
- x = 4, y = 6 ✓
- x = 10, y = 3
- x = 7, y = 3

Misconceptions:

- Non comprendere l'ordine di esecuzione delle operazioni e il loro effetto sulle variabili.
- Presumere che il valore finale di una variabile sia indipendente dalle operazioni precedenti.

MC1: Qual è il nome del metodo chiamato da “x”?

```
int main() {  
    int x = 5;  
    x = incrementa(x); // Chiamata della funzione  
    printf("Il valore di x è: %d\n", x); // Stampa del valore aggiornato  
    return 0;  
}  
  
// Definizione della funzione  
int incrementa(int a) {  
    a = a + 3; // Incrementa il valore di a di 3  
    return a; // Ritorna il valore aggiornato  
}
```

Risposta: incrementa.

Misconceptions:

- Confondere la definizione del metodo con la sua invocazione.

MC2: Quale dei seguenti comandi di stampa è corretto?

int x = 2;

- printf("%d", x);
- printf("%d", 2);
- printf("%s", x);
- printf(x);

Misconceptions:

- Credere che una funzione possa essere chiamata senza passare argomenti, anche se ne richiede. La stampa senza argomenti avrà come risultato come la stampa di uno spazio vuoto.

LV1 : come FORS1

AR1 : Cosa significa passare un argomento a un metodo?

- Dare un valore al metodo da usare.
- Restituire un valore dal metodo.
- Creare un nuovo metodo.
- Chiamare il metodo.

Misconceptions:

- Confondere gli argomenti con i valori di ritorno.

TA1: Qual è il tipo dell'argomento del metodo chiamato?

```
char s1[] = "Mary";
char s2[] = "Marianne";
int x;

x = strcmp(s1, s2);
```

Risposta: int

Misconceptions:

- Confondere il tipo dell'argomento con il tipo del valore di ritorno.

Consegna 5

Consegna eseguita in classe.

PREREQUISITI

- Conoscenza di base dei concetti di sequenza:
 - Saper ordinare semplici azioni in modo corretto (es. "prima mi lavo i denti, poi faccio colazione").
- Concetto di simbolo e associazione significato-simbolo:
 - Riconoscere che un oggetto o colore può rappresentare un'informazione (es. "il rosso significa stop").
- Esempi di scomposizione di problemi nella vita quotidiana:
 - Risolvere compiti semplici dividendo il lavoro in piccoli passi (es. per apparecchiare la tavola: "prendo il piatto", "prendo la posata", ecc.).
- Capacità di seguire una legenda:
 - Saper associare correttamente simboli o colori a significati predefiniti utilizzando una guida (es. "cerchio verde = avanti").

TRAGUARDI:

- TP4: spiega usando il ragionamento logico perché un programma strutturalmente semplice raggiunge i suoi obiettivi -> raggiunto durante la costruzione e la spiegazione di una legenda;
- TP5: inizia a riconoscere la differenza tra l'informazione e i dati -> raggiunto sempre nella legenda e nel suo utilizzo quando si scinde tra i dati considerati e le variabili della legenda;
- TP6: esplora la possibilità di rappresentare dati di varia natura (numeri, immagini, suoni, ...) mediante formati diversi, anche arbitrariamente scelti -> raggiunto con l'utilizzo di elementi diversi per concetti simili (fagioli usati in maniera simile alla pastina);

OBBIETTIVI:

- Per la classe terza:

- O-P3-A-2: comprendere che problemi possono essere risolti mediante la loro scomposizione in parti più piccole
 - O-P3-P-2: ordinare correttamente la sequenza di istruzioni;
 - O-P3-P-4: utilizzare la selezione ad una via per prendere decisioni all'interno di programmi semplici -> raggiunto attraverso l'utilizzo di più elementi uguali usati come limitatori ;
 - O-P3-D-1: scegliere ed utilizzare oggetti per rappresentare informazioni familiari semplici (es. colori, parole, ...) -> toccato attraverso l'utilizzo di elementi di vita quotidiana,
 - O-P3-D-2: definire l'interpretazione degli oggetti utilizzati per rappresentare l'informazione (legenda).
- Per la classe quinta:
 - O-P5-A-2: risolvere problemi mediante la loro scomposizione in parti più piccole;
 - O-P5-P-3: riconoscere che una sequenza di istruzioni può essere considerata come un'unica azione oggetto di ripetizione o selezione;
 - O-P5-D-1: utilizzare combinazioni di simboli per rappresentare informazioni familiari complesse (es. colori secondari, frasi, ...);