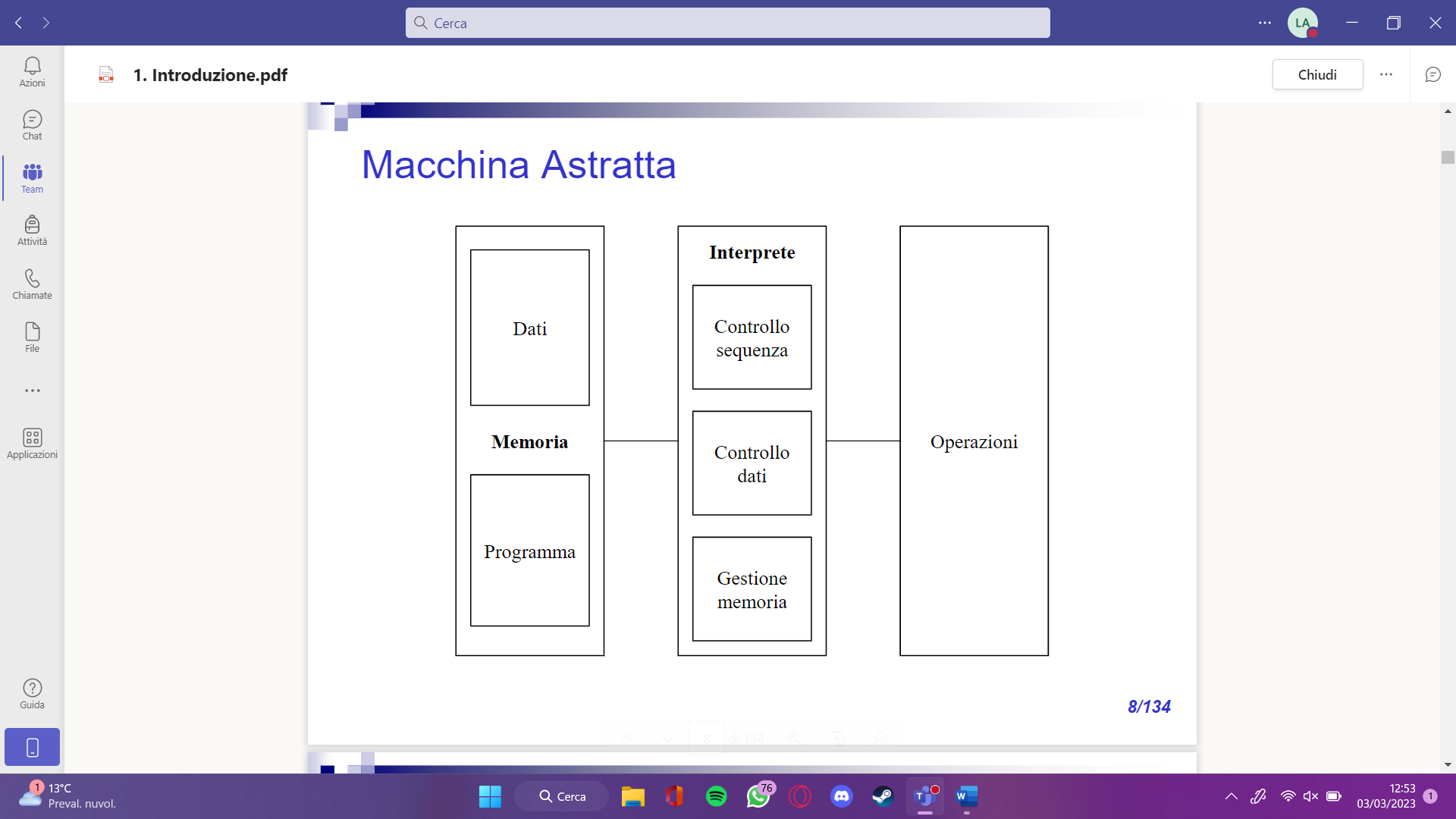
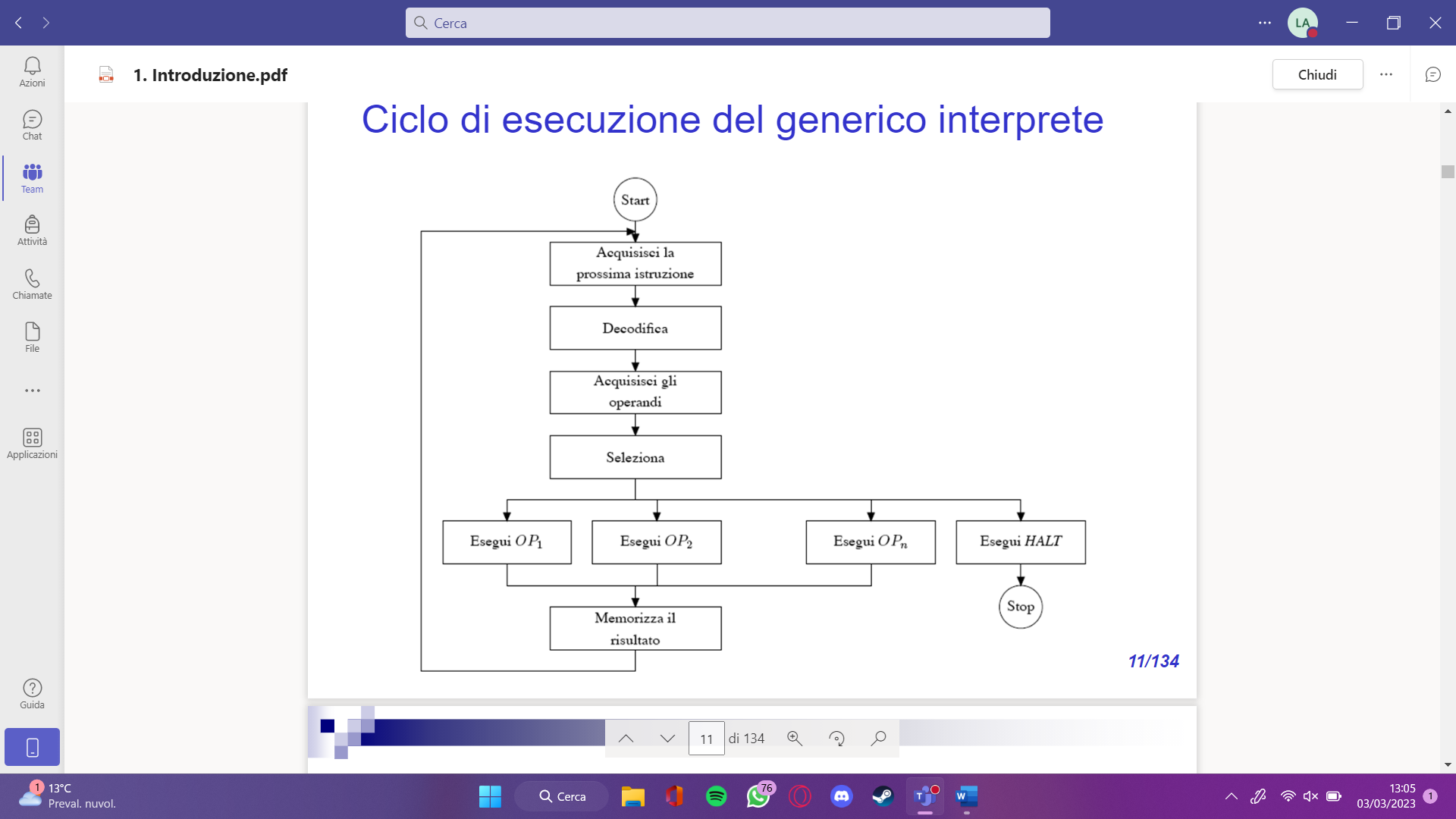
**rimo giorno linguaggi:  
-Lingue naturali**: Hanno analogie e le somiglianze derivanti dalla genealogia  
 Es: Cioccolata = ciocolata, insalta = salata, treno =tren.  
**-Linguaggi di programmazione**: è difficile conoscere un gran numero in modo approfondito  
 -E’ possibile conoscere a fondo i meccanismi che ispirano il progetto e **l’implementazione.** -E’ difficile conoscerne un gran numero in modo approfondito.  
**Macchina Astratte e implementazione dei linguaggi di programmazione**  
Qual è il significato dell’espressione: **implementazione di un linguaggio di programmazione?**  
Il concetto è strettamente collegato a quello delle **macchine astratte**  
**Calcolatore**: macchina fisica e inoltre:   
- Consente di eseguire algoritmi formalizzati(tradotti) perché siano “comprensibili” dall’esecutore.  
 - **Formalizzazione**: codifica di algoritmi in un certo linguaggio (**L**) definito da una specifica sintassi.  
 - La sintassi di **L** permette di utilizzare i costrutti per comporre programmi in **L**.   
- Un programma in L è una sequenza di istruzioni del linguaggio **L**.  
**Macchina astratta**: Non è ben definita e avviene l’astrazione del concetto di calcolatore fisico (è come il calcolatore ma senza hardware associato (astratta)). -solo software.   
**Macchina astratta per L:**-Un insieme di algoritmi e strutture dati che permettono di eseguire programmi scritti in **L**.  
-Questa macchina è denotata come: **ML.**  
Composta da :  
- **Una memoria** per immagazzinare dati e programmi  
- **Un interprete** per interpretare ed eseguire le istruzioni contenuti nei programmi.  
**Linguaggio Macchina**:  
Se abbiamo una macchina astratta **ML**, il linguaggio **L** “compreso” dall’interprete di **ML** è detto linguaggio macchina di **ML.**  
**Macchina Astratta:**  
**Interprete:**Se siamo in presenza di un **L**, l’interprete è un programma che si occupa del controllo della sequenza delle istruzioni del linguaggio, controllo dei dati e gestire la  
memoria, es: va a prendere i valori delle variabili dalla memoria.  
**Operazioni dell’interprete:  
1. Operazioni per elaborazione dei dati primitivi:**- Numeri interi, reali.  
- Operazioni aritmetiche.  
**2. Operazioni e le strutture dati per il controllo della sequenza di esecuzione:**- Gestiscono il flusso di controllo delle istruzioni.  
- Strutture dati per memorizzare l’indirizzo della prossima istruzione  
- Operazioni per manipolare le strutture dati, Es: Calcolo indirizzo prossima istruzione.  
**3. Operazioni e strutture dati per il controllo del trasferimento dei dati:**- Gestisco il trasferimento dei dati dalla memoria all’interprete e viceversa (Sia dati che indirizzi).  
Es: recupero gli operandi  
- Possono far uso di strutture dati ausiliarie  
 es: pila.  
**Operazioni e strutture dati per la gestione della memoria:**-allocazione della memoria per i dati e programmi.  
  
**Traduzione foto:** Acquisisce la prossima istruzione -> Decodifica(cerca di capire che tipo di istruzione è), in base all’istruzione acquisisce degli operandi e successivamente seleziona il tipo di operazione e la esegue, dopo memorizza il dato e ritorna all’inizio del ciclo, non ha termine fino a quando il programma è terminato(END).  
  
 **Realizzazione della macchina astratta:  
1.** Una **ML** per essere utilizzata deve prima o poi utilizzare qualche dispositivo fisico.  
Realizzazione “fisica” in hardware:  
 - Algoritmi di **ML** realizzati mediante dispositivi fisici.  
**2.** Realizzazioni che usano livelli intermedi tra **ML** e dispositivo fisico.  
- **Simulazione mediante software.**  
-Emulazione mediante firmware(microprogrammi in linguaggi di basso livello).  
**Firmware:** Hardware programmabile(eprom).(ne software ne hardware)  
**REALIZZAZIONE MEDIANTE SOFTWARE:**  
- Algoritmi e strutture dati di **ML** realizzati in un altro linguaggio **L’** già implementato.  
- **ML** è realizzata mediante programmi **L’** che simulano le funzionalità di **ML**.  
- **ML** è realizzata attraverso la macchina **M’L’.**  
- **M’L’** è detta macchina ospite denotata con **MoLo.**- L’implementazione di **L** sulla macchina ospite avviene mediante una qualunque “**traduzione”** di **L** in **Lo.**A seconda della traduzione si parla di:  
- **Implementazione interpretativa.**- **Implementazione compilativa.**  
**FUNZIONI PARZIALI:  
Una funzone parziale**  
  
  
è una corrispondenza tra elementi dell’insieme A e quelli dell’insieme B.  
- **Parziale**: La corrispondenza può essere non definita per qualche elemento di A.  
- Dato a A, può esistere un corrispondente in B,denotato con f(a) altrimenti non esiste.  
I programmi definiscono funzioni parziali:  
read(x);  
if x==1 then print(x) else   
while (x<>1) do skip;  
  
  
**Definizione di interprete:**  
Un programma scritto in linguaggio **L** si può vedere come una funzione parziale:   
  
Possiamo definire la seguente definizione di interprete di **L** in **Lo:**



Non vi è traduzione esplicita dei programmi scritti in **L,** solo il procedimento di decodifica.  
L’interprete per eseguire l’istruzione **I** di **L,** le fa corrispondere ad un insieme di istruzioni in **Lo.** Tale decodifica non è una traduzione esplicita poiché il codice a **i** è eseguito direttamente e non prodotto in uscita.

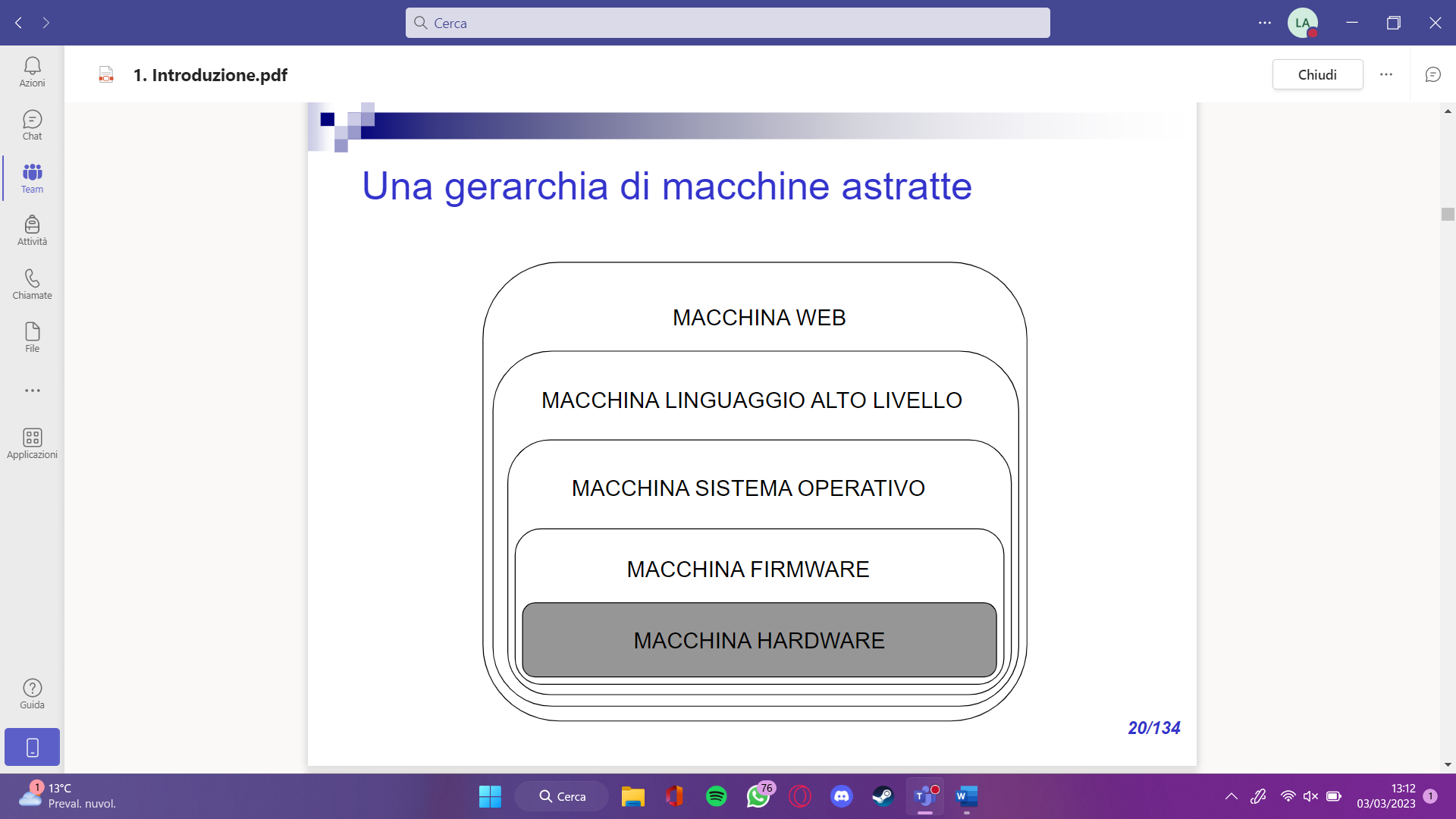
L’interprete funziona bene se preso l’ingresso del programma è in grado di restituirmi l’output.  
arriva fino a pag 19 a scrivere e studiare. **DEFINIZIONE DI COMPILATORE**  
Un compilatore da **L** a **Lo** è un programma che realizza la funzione:



Traduzione esplicita dei programmi scritti in **L** in programmi scritti in **Lo**  
  
  
Per eseguire **P** su Input, bisogna eseguire CL,Lo con **P** come input.  
Si avrà come risultato un programma compilato PC scritto in **Lo,** che sarà eseguito su **MoLo** con il dato in ingresso Input.  
**Interpretazione vs Compilazione:  
-Interpretazione  
Scarsa efficienza:**  
 -decodifica costrutti a tempo di esecuzione.  
 -decodifica di uno stesso comando ripetuto.  
**Maggiore flessibilità:**  
 -debugging più semplice.  
**-Compilazione:  
Maggiore efficienza:** -decodifica di un’istruzione fatta una sola volta indipendentemente da quante volte è eseguita.  
**Minore flessibilità:** -perdita di informazioni rispetto al programma sorgente(difficile tracciare errori e dunque maggiori difficoltà nel debugging).



**GERARCHIE DI MACCHINE ASTRATTE**Pensiamo ad una gerarchia di macchine astratte come: **ML0,ML1,ML2,…MLn.  
MLi** è implementata sfruttando il linguaggio della macchina sottostante **MLi-1.  
MLi** fornisce a sua volta il proprio linguaggio alla macchina sovrastante **MLi+1.**Indipendenza fra livelli:Modifiche interne alla funzionalità di un livello non hanno influenza sugli altri livelli.  
**Es gerarchia di macchine astratte:**

 **Linguaggi di programmazione**: è un linguaggio formale che ci consente di portare su un livello di macchina fisica l’algoritmo di soluzione**(Ap)** per un problema **P**.  
Per portare l’algoritmo di risoluzione a livello di macchina fisica per il problema si fa riferimento alla gerarchia di macchina astratta, traducendo dal livello n al livello più basso fino alla macchina hardware.  
Immagine che contiene testo

Descrizione generata automaticamente  
**1-** Esistono problemi per i quali non esiste un programma che li risolva?  
**2-** La risposta dipende dal linguaggio di programmazione?  
  
  
  
  
  
**FUNZIONE (Parziale) CALCOLABILE:**  
Definire quando una funzione parziale è calcolabile o meno, **f:A->B** **è calcolabile** in un linguaggio **L** se esiste un programma **P** scritto in **L** tale che:  
- Se f(a)=b allora **P** quando gli do in **input a**, termina e produce come **output b**, associato dalla funzione f al valore a.  
- Se f(a) essendo parziale non è definita allora il **P** con **input a** non deve terminare ma deve andare in ciclo all’infinito.  
Con questa definizione rispondiamo alla prima domande e possiamo individuare una classe di problemi per la quale non esiste un programma che possa risolvere il problema.  
**Esistono problemi non calcolabili ? Il problema della fermata**:  
Vogliamo stabilire se dato un programma **P** e dato un input del programma, termina oppure clicla.  
Supponiamo l’esistenza di un programma di debugging H, prende in ingresso **P** scritto in liguaggio **L** e il secondo **parametro di input x** per stabilire se **P** termina o no, **H** restituirà **true se P(x) termina , false se P(x) va in loop.**Boolean H(P,x)  
 booleanterm;  
 if (P(x) termina) then term=true;  
 else term=false;  
 return term;  
  
Possiamo costruire un **altro programma K** scritto in **L** che prenda in input un programma  
**P** scritto in **L .**  
**Il programma** **K sfrutta H per decidere sulla terminazione di P.**  
K(P)  
if (H(P,P)=false) then print(“LOOP”);  
 else while (true) do print(“TERMINA”);  
  
  
Quando il programma P ha in ingresso un dato che è la rappresentazione dello stesso programma P se è false, allora stampa (loop), altrimenti print(termina).  
Qindi la terminazione di K è opposta all’input (P).  
Se a K dessi in ingresso K, avremo o Stampa “LOOP” se K(K) non termina o va in loop se K(K) termina.  
**Dimostrazione per assurdo:** assurdo dall’esistenza del programma in K.  
- K(K) termina con una stampa quando K(K) non termina   
- K(K) non termina quando K(K) termina  
-Assurdo conseguente dall’aver supposto l’esistenza del programma H  
**Quindi esistono funzioni non calcolabili? SI:**  
Abbiamo dimostrato l’esistenza di problemi, funzioni non calcolabili(indecibili).  
  
**2:La risposta dipende dal linguaggio di programmazione?**   
Risposta data da un modello di calcolo matematico che possiamo pensare come il primo linguaggio, **Macchine di Turing(Mdt).**  
Primo formalismo (=linguaggio) per implementare tutti i programmi in c, phyton ecc… ci da la possibilità di scrivere in qualsiasi linguaggio.  
Con le MdT è stata dimostrata l’indecibilità del problema della fermata.  
**Formalismo Mdt:** una macchina non necessario da realizzare fisicamente, astratta e costituita da un unità di controllo con l’algoritmo di risoluzione per la classe di problemi.  
**Com’è costituita la MdT:  
- L’unità di controllo:** la quale decodifica ed esegue i comandi rivolti alla testina (al suo interno troviamo l’algoritmo)  
**- Testina di lettura e scrittura**: Può leggere/scrivere e spostarsi a destra o sinistra di una cella per volta e gestisce un controllo espresso da un numero finito di stati.  
**­- Nastro:** diviso in modo infinito in celle (memoria) dove in ogni cella si trova un simbolo preso da un alfabeto finito, in questo caso nella figura es: (0,1)

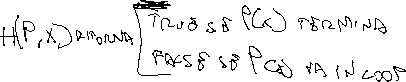
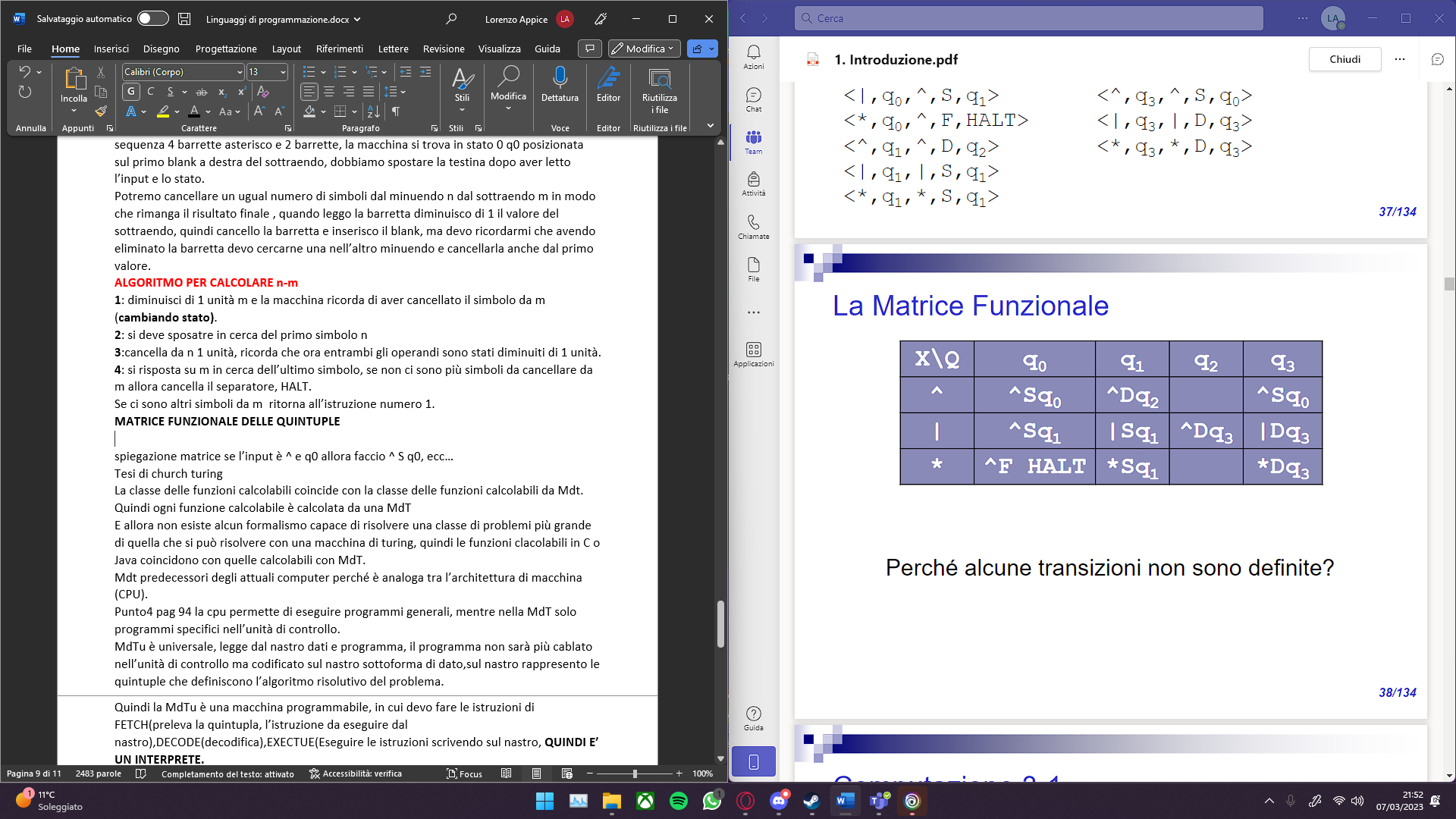


Immagine che contiene testo

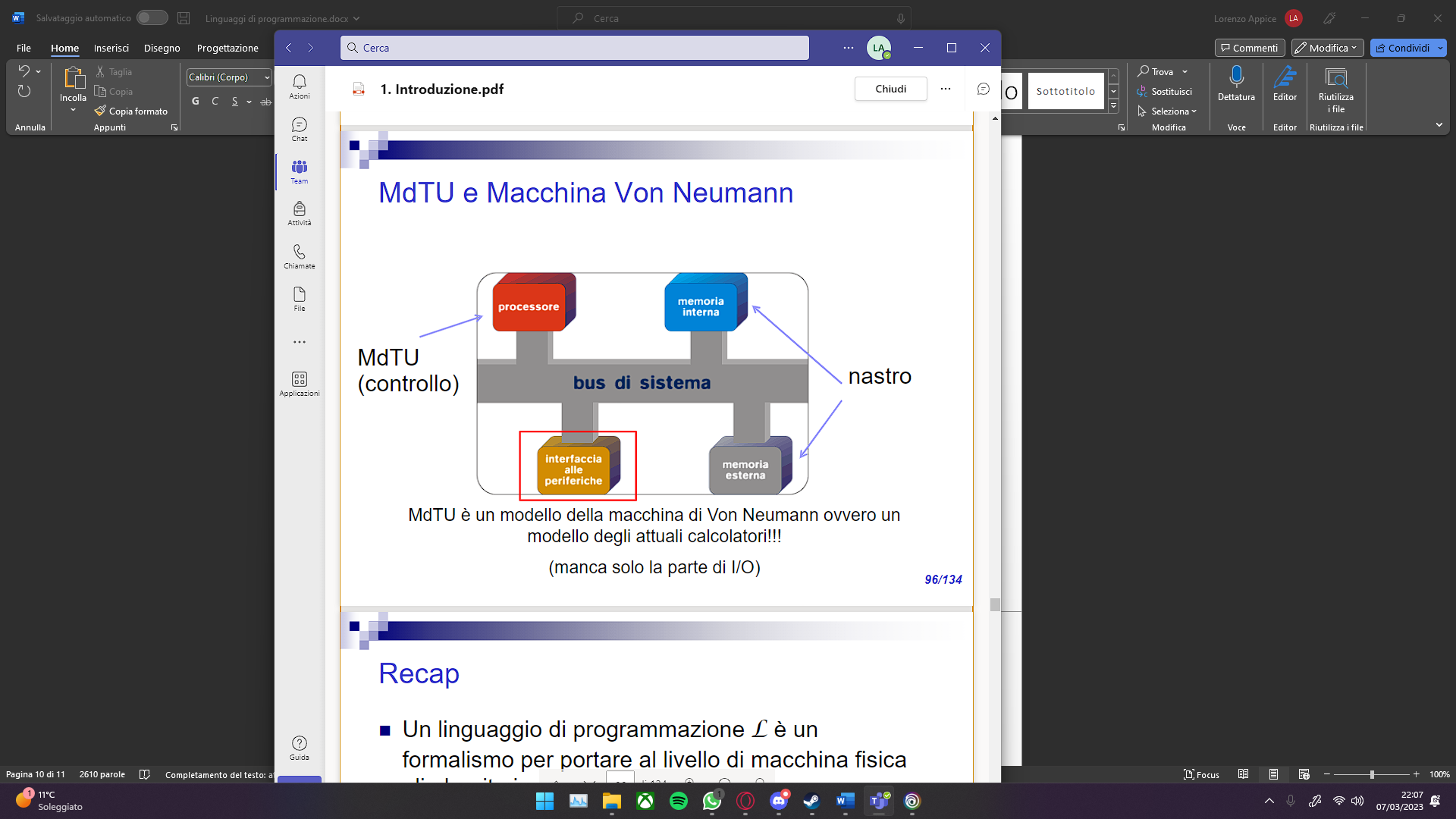
Descrizione generata automaticamente  
**L’unità di controllo**: può avere differenti stati (una cosa simile è il cambio di marcia dell’auto, prima,seconda ecc..), determinano un comportamento differente della macchina.  
Quindi esegue un programma **P** sui dati memorizzati sul nastro.  
**Le istruzioni essenziali di input/output sono 5**, racchiuse dalle parentesi angolari:   
< simbolo\_letto, è il simbolo che leggiamo attraverso la testina (in questo caso 0).  
 stato\_corrente, stato dell’unità di controllo.  
 simbolo\_da\_scrivere, il nuovo simbolo che la macchina scriverà dove il simbolo era letto.  
 sinistra/destra, la testina si sposta da sinistra a destra dando l’output per lo spostamento.  
 nuovo\_stato > possiamo prevedere lo stato dell’unità di controllo.  
L’ordine di questi 5 elementi è importante, perché il primo e il secondo elemento sono dati di input , il terzo quarto e quinto di output.  
 **Una MdT è definita quindi cosi (da una quintupla):  
M = (X,Q,fm,fd, delta).**  
**X**= insieme finito di simboli, (Es:0 e 1),le celle vuote rappresentate da un simbolo chiamato blank.  
**Q**= Insieme finito di stati,lo stato “HALT” che definisce la terminazione.  
**Funzione fm(funzione di macchina):** è dato dal prodotto di **Q \* X -> X** per calcolare un valore.  
Deve avere in ingresso 2 elementi, lo stato e il simbolo di X, quando ha questa coppia decide qual è il simbolo da scrivere sul nastro.  
**Funzione fd (stabilisce la direzione):** a seconda del prodotto **di Q x X-> {S,D,F}**. determina lo spostamento S,D,F (Sinsitra,destra, fermo).  
**Funzione Delta:** Coppia Q x X ->Q mi consente di cambiare stato.  
Definisce il nuovo stato, chiamato anche stato successivo.  
**SCRIVERE ALGORITMI PER MdT: Nastro**  
Immagine che contiene testo

Descrizione generata automaticamente  
**X è la barretta verticale**, 4 barrette verticali denotano il primo operando, le altre due barrette verticali rappresenteranno il sottraendo, **X un altro simbolo (separatore) “\*”** **e poi il simbolo di blank “^”.**  
Se ha funzionato correttamente, dovrebbe terminare (stato di HALT) lasciare sul nastro l’output corretto:  
Immagine che contiene testo

Descrizione generata automaticamente  
**La sottrazione invece** è un’operatozione, è sempre una funzione quindi non si trova tra le operazioni native della macchina , non sa fare il meno, (non è come la calcolatrice) quindi bisogna implementarla scrivendo il programma/algoritmo della sottrazione (con il formalismo delle macchine di turing).  
**SCRIVERE ALGORITMI PER MdT: Controllo**  
**-Definire le funzioni fm, fd e delta**.  
**-**Il programma per una MdT è una sequenza di multiple:  
 <xi∈X, qj∈Q, **xij∈X**, **{S,D,F}, qij∈Q**>  
**xij=fm(xi,qj) nella cella corrente**  
**Spostamento della testina fd(xi,qj)  
nuovo stato qij= Delta(xi,qj)**  
  
  
  
  
  
  
  
  
  
  
**SCRIVERE ALGORITMI PER MdT: Sottrazioen tra interi**  
Immagine che contiene testo

Descrizione generata automaticamente  
Progettare l’algoritmo per la sottrazione tra 2 numeri interi n e m >=0,il nastro con la sequenza 4 barrette asterisco e 2 barrette, la macchina si trova in stato 0 q0 posizionata sul primo blank a destra del sottraendo, dobbiamo spostare la testina dopo aver letto l’input e lo stato.  
Potremo cancellare un ugual numero di simboli dal minuendo n dal sottraendo m in modo che rimanga il risultato finale , quando leggo la barretta diminuisco di 1 il valore del sottraendo, quindi cancello la barretta e inserisco il blank, ma devo ricordarmi che avendo eliminato la barretta devo cercarne una nell’altro minuendo e cancellarla anche dal primo valore.  
**ALGORITMO PER CALCOLARE n-m**  
**1**: diminuisci di 1 unità m e la macchina ricorda di aver cancellato il simbolo da m (**cambiando stato)**.  
**2**: si deve sposatre in cerca del primo simbolo n   
**3**:cancella da n 1 unità, ricorda che ora entrambi gli operandi sono stati diminuiti di 1 unità.  
**4**: si risposta su m in cerca dell’ultimo simbolo, se non ci sono più simboli da cancellare da m allora cancella il separatore, HALT.  
Se ci sono altri simboli da m ritorna all’istruzione numero 1.  
**MATRICE FUNZIONALE DELLE QUINTUPLE**X={I, \*, ^} Q={q0,q1,q2,q3,HALT}  
q0= stato iniziale della computazione ovvero ricerca ultimo simbolo di m  
q1= diminuito m  
q2≡ raggiunto simbolo iniziale di n  
q3≡ diminuiti entrambi operandi  
Immagine che contiene testo

Descrizione generata automaticamente  
spiegazione matrice: se l’input è ^ e q0 allora faccio ^ S q0, ecc…  
**Es Computazione 3-1 da pag 39 a pag 62 pdf introduzione  
Es Computazione 1-1 da pag 63 a pag 81 pdf introduzione  
Es Computazione 2-0 da pag 82 a pag 86 pdf introduzione  
Es Computazione 0-0 da pag 81 a pag 91 pdf introduzione  
Esercitazioni: 1:**Stabilire una stringa binaria contiene lo stesso numero di ‘0’ e ‘1’   
**2:** Stabilire se una stringa binaria è palindroma (ovvero si legge indifferentemente da S a D, es.: 01000010)  
**3:** Stabilire se un numero rappresentato con ‘|’ è pari oppure dispari  
**TEST DI CHURCH-TURING**  
La classe delle funzioni calcolabili coincide con la classe delle funzioni calcolabili da Mdt.  
Quindi ogni funzione calcolabile è calcolata da una MdT  
Non esiste alcun formalismo capace di risolvere una classe di problemi più grande di quella che si può risolvere con una macchina di turing, le funzioni clacolabili in C o Java non sono di più della Mdt ma a sua volta coincidono con quelle calcolabili con MdT.  
Mdt è predecessore degli attuali computer perché è analoga all’architettura di macchina (CPU).  
Immagine che contiene testo

Descrizione generata automaticamente  
La cpu permette di eseguire programmi generali, mentre la MdT solo programmi specifici nell’unità di controllo.  
Per questo è stata introdotta la **MdTu** (che è universale): legge dal nastro i dati e il programma, quindi non sarà più cablato nell’unità di controllo ma codificato sul nastro sottoforma di dato.  
Sul nastro sono rappresentate quindi le quintuple che definiscono l’algoritmo risolutivo del problema.  
**La MdTu** è una macchina programmabile, in cui devo fare le istruzioni di FETCH(preleva la quintupla(istruzioni) da eseguire dal nastro),DECODE(decodifica),EXECTUE(Eseguire le istruzioni scrivendo sul nastro, **QUINDI E’ UN INTERPRETE.**  
Recap:  
Un linguaggio di programmazione **L** è un formalismo per portare al livello di macchina fisica gli algoritmi , implementare **L** significa realizzarne l’interprete ovvero il programma che traduce **L** nel linguaggio di macchina ospite.  
-La possibilità di risolvere un problema non è dovuto al linguaggio di programmazione dato che tutti i linguaggi possono calcolare esattamente le funzioni calcolate dalle MdT.  
-Quindi tutti i linguaggi di programmazione si dicono Turing-Completi.  
Capitoli 1 e 5 dal libro Maurizio Gabbrielli, Simone Martini. Linguaggi di Programmazione: Principi e paradigmi. Seconda edizione. McGraw-Hill, 2011.  
  
**LA TEORIA DEI LINGUAGGI FORMALI**  
Livelli di descrizione di un linguaggio:  
-Grammatica : Quali sono le frasi corrette?  
-Semantica: Cosa significa una frase corretta?  
-Pragmatica: Come usare una frase corretta e sensata?  
-implementazione (per i linguaggi di programmazione): Come eseguire una frase corretta in modo da rispettarne il significato?  
**Concetto intuitivo di gramamtica**:  
Alfabeto, lessico:parole del linguaggio ecc.. pag 101  
Grammatica 2 tipi:  
- Regolari ;  
- Libere da contesto; pag 102  
L’informatica scienza degli algoritmi: elaboratori,informazione,programmi  
3 brance principali dell’informatica teorica:  
-Teoria della computabilità: problemi che so legare agli algoritmi, modelli di computazione(calcolo) quindi una classe di problemi presuppone che io possa risolverli con algoritmi di risoluzione ma prima devo stabilire il modello di calcolo con cui devo progettare l’algoritmo.  
nata con matematici e poi per interesse per gli informatici.  
tra i modelli di computazione studieremo la teoria degli automi si sovrappone con la teoria dei linguaggi formali.  
-Teoria della complessità computazionale:  
-Teoria matematica della computazione:

