DOMANDA 1: Consideriamo un linguaggio, L, come si specificano il lessico di L e la sintassi di L?
Il lessico noto anche come vocabolario o insieme di simboli, è l'insieme di parole chiave, identificatori e simboli speciali utilizzati per scrivere programmi in quel linguaggio. Ad esempio, in Java le parole chiave sono "int", "while", "for", mentre gli identificatori sono i nomi dati alle variabili e alle funzioni.
La sintassi di un linguaggio descrive le regole grammaticali per la scrittura di programmi in quel linguaggio. Ad esempio, in Java la sintassi specifica come le dichiarazioni delle variabili devono essere scritte, come le funzioni devono essere definite e chiamate, e come i controlli di flusso devono essere utilizzati per controllare il flusso del programma. La sintassi può essere descritta utilizzando diagrammi di flusso, grammatiche formali o esempi di codice.

DOMANDA 2: Cosa si fa durante
1)L'Analisi Lessicale
2)L'Analisi Sintattica e
3)L'Analisi Semantica (controllo dei tipi)
Quali sono gli input e gli output delle tre analisi precedenti?

L'analisi lessicale è eseguita dallo scanner che legge l'istruzione e genera un token e lo passa al parser. Dunque, lo scanner scompone il programma in token.
Lo scanner (quindi l'analizzatore lessicale) viene rappresentato da un automa a stati finitideterministici.
è una fase del processo di compilazione che si occupa di analizzare il codice sorgente e di riconoscere i simboli del linguaggio: il codice sorgente viene letto carattere per carattere e vengono generati una serie di "token" (unità significative del linguaggio), che rappresentano i simboli del linguaggio come parole chiave, identificatori, costanti e simboli speciali. é responsabile di riconoscere e generare questi token, utilizzando un insieme di regole predefinite per ciascun simbolo, gestire inoltre commenti e spazi bianchi.
Esempio:
Input newVal = oldVal + 1 - 3.2
Output ⟨id : newVal⟩⟨assign⟩⟨id : oldVal⟩⟨plus⟩⟨intNum : 1⟩⟨minus⟩⟨floatNum : 3.2⟩
L'input dell'analisi lessicale è il codice sorgente l'output è il token appena letto.

L'analisi Sintattica nota anche come parsing è una fase del processo di compilazione che si occupa di verificare che il codice sorgente segua le regole grammaticali del linguaggio: il compilatore utilizza i token generati dall'analisi lessicale come input e verifica che essi siano combinati correttamente per formare frasi valide del linguaggio.
In caso di errori sintattici, il compilatore può generare un messaggio di errore per indicare la posizione e la natura dell'errore, in modo che il programmatore possa correggere il codice.
Il parser organizza i token in un parse tree e inoltre, riconosce quali produzioni sono state usate per generare la sequenza di Token.
Il parse tree è l'albero di derivazione della grammatica. Nel nostro caso il parse tree generato è un ast, ossia un parse tree semplificato, con meno informazioni (con meno nodi). L'ast ricostruisce la grammatica lineare del codice.
Esistono due tecniche di parsing:
TOP-DOWN: In questo caso si costruisce l'ast a partire dalle grammatiche LL e si costruisce l'albero partendo dalla radice andando verso e foglie.
BOTTOM-UP: L'ast si costruisce a partire dalla grammatica LR e si costruisce l'albero partendo dalle foglie e proseguendo verso la radice.
L'input dell'analisi sintattica sono i token mentre l'output è l'ast.

L'analisi semantica è una fase del processo di compilazione che si occupa di verificare il significato del codice sorgente e di controllare che sia corretto in base alle regole del linguaggio.
Il compilatore utilizza l'albero di sintassi generato durante l'analisi sintattica come input e verifica che il codice sorgente sia semantica e logicamente corretto:
Verifica delle dichiarazioni di tipo: il compilatore controlla che i tipi di dati siano compatibili tra loro e che le variabili siano dichiarate (una sola volta) prima di essere utilizzate.
Verifica dei valori delle espressioni: il compilatore controlla che le espressioni siano valide e che i valori restituiti siano compatibili con i tipi di dati dichiarati (All'occorrenza si effettua una conversione)
Controllo dei limiti: il compilatore verifica che i valori di una variabile siano compresi nell'intervallo consentito per quel tipo di dati.
La raggiungibilità del codice: verifica che non ci sia codice irraggiungibile o codice "morto"
Come output ho di nuovo l'AST già utilizzato ma viene verificato che il modo in cui sono organizzati i nodi è corretto: in caso di errori semantici, il compilatore può generare un messaggio di errore per indicare la posizione e la natura dell'errore, in modo che il programmatore possa correggere il codice.

DOMANDE SULLO SCANNER

DOMANDA 1: Quali classi lessicali fanno in generale parte di un linguaggio di programmazione?
Ogni elemento del lessico appartiene ad una delle seguenti sei classi lessicali:
1 - PAROLE CHIAVE: non possono essere usate come identificatori parole come (for, if, else ...)
2 - DELIMITATORI: (; {))
3 - OPERATORI: (*, +, -, …)
4 - COMMENTI
5 - IDENTIFICATORI
6 - COSTANTI
Le prime quattro sono classi lessicali chiuse, e i loro elementi sono composti solo dal loro nome.
Le ultime due sono classi lessicali aperte, ed i loro elementi sono composti da nome e da attributo semantico.
Le classi lessicali sono rappresentate da espressioni regolari.

DOMANDA 2: Cosa è un Token?
Un Token descrive un insieme di caratteri che hanno lo stesso significato come ad esempio, identificatori, operatori, keywords, numeri, delimitatori. Le espressioni regolari sono usate per descrivere i token. Il token è il risultato dell'analizzatore lessicale e viene dato come input all'analizzatore sintattico.

DOMANDA 3: Fare esempi di Token?

Token	Pattern	Classe rappresentata
INT	[1-9][0-9]* | 0	Costante
FLOAT	([1-9][0-9]* | 0).[0-9]{1,5}	Costante
ID	[a-z]+	Identificatore
TYINT	int	Parola chiave
TYFLOAT	float	Parola chiave
ASSIGN	=	Operatore
PRINT	print	Operatore
PLUS	+	Operatore
MINUS	-	Operatore
SEMI	;	Delimitatore
EOF	(char) -1	Fine Input

DOMANDA 4: Come (con quale classe di linguaggi) si specificano i token dei linguaggi?
Per specificare i token viene utilizzate grammatiche formali; queste grammatiche descrivono le regole grammaticali del linguaggio di programmazione, includendo le regole per la generazione dei token. Ad esempio, una regola potrebbe specificare che un identificatore è composto da una lettera seguita da una serie di lettere, numeri o trattini bassi.

DOMANDA 5: Come si può implementare il riconoscimento lessicale?
Il riconoscimento lessicale può essere implementato attraverso l'ausilio di automi a stati finiti: un automa a stati finiti è una macchina formale che riconosce un linguaggio regolare e può essere utilizzato per riconoscere i token in un compilatore, passando attraverso gli stati dell'automa a seconda dei caratteri del codice sorgente (letti in input all'automa)

DOMANDE PARSING

DOMANDA 1: Definizione di grammatica LL(1)?
Una grammatica è LL(1), se per ogni simbolo non-terminale A, un token predice al più unaproduzione. Cioè una volta fatta la tabella Predict per ogni simbolo non terminale A:
Se le produzioni, p_1, ...,p_n associate ad A e $Pred_1$, ,$Pred_n$ sono gli insiemi predict associati a p_1, ...,p_n, allora $Pred_i \cap Pred_j = \emptyset$ per tutti $1 \leq i$ $j \leq n$.
Un linguaggio è LL(1), se ha una grammatica LL(1) che lo genera.

DOMANDA 2: Per quali ragioni una grammatica può non essere LL(1)?
Una grammatica ricorsiva sinistra non può essere LL(1). Ad esempio, E → E + T | E − T | vPerché? Assumi A → A α | β
→Se a ∈ First(β) allora a ∈ First(A α), perché A ⇒ A α ⇒ β α
→E β = ϵ allora a ∈ Follow(A) se a ∈ First(α) e quindi anche a ∈ First(A α), perché A ⇒ Aα⇒ α Una grammatica con prefissi comuni, cioè 2 o più produzioni per lo stesso non terminale hanno lastessa parte iniziale, non può essere LL(1).
Ad esempio, S → if E then E | if E then E else E.Perché?
→se A → α β1 | α β2, se a ∈ First(α β1) allora a ∈ First(α β2), e viceversa.
Ma il linguaggio generato può essere LL(1), se troviamo un'altra grammatica LL(1) che lo genera

DOMANDA 3: Come si può specificare un parser Top-Down?
Un parser Top-Down può essere specificato utilizzando l'algoritmo di parsing a discesa ricorsiva, per specificare un parser Top-Down utilizzando l'algoritmo di parsing a discesa ricorsiva, è necessario seguire i seguenti passi:

1. Specificare la grammatica del linguaggio in forma normale di Chomsky.
2. Scrivere una funzione per ogni non terminale della grammatica che implementi la logica di parsing per quel non terminale.
3. Utilizzare le funzioni dei non terminali per generare l'albero sintattico, chiamando le funzioni appropriate in base all'input e allo stato corrente del parser.

DOMANDE: AST e Symbol Table

DOMANDA 1: Cosa è ed a cosa serve definire un Abstract Syntax Tree per una grammatica di un linguaggio di programmazione?
Un albero sintattico astratto (AST) è una rappresentazione astratta della struttura sintattica di un programma scritto. Serve per rappresentare la struttura logica di un programma in modo semplice e indipendente dalla sintassi esatta del linguaggio di programmazione.

DOMANDA 2: Quale è la differenza fra AST e parsing tree per una stringa di un linguaggio di programmazione?
l'AST è una rappresentazione più astratta e semplificata della struttura sintattica del programma, che elimina i dettagli di formattazione e sintassi del codice sorgente e si concentra solo sulla struttura logica del programma. Il parsing tree è invece una rappresentazione esatta della struttura sintattica del programma che segue esattamente la grammatica del linguaggio di programmazione.

DOMANDA 3: Cosa è la Symbol Table, quali informazioni contiene e a cosa serve? Come può essere implementata una Symbol Table?
Una tabella dei simboli mantiene l'associazione fra gli identificatori e le informazioni ad essi associati(tipo, definizione, ecc.). È condivisa dalle varie fasi della compilazione serve per fornire un meccanismo per l'accesso veloce alle informazioni sui simboli utilizzati all'interno del codice, rendendo più facile per gli strumenti di elaborazione del codice sorgente analizzare e modificare il codice in modo efficace.
Per implementarla si una tabella hash per associare i nomi dei simboli alle relative informazioni.

DOMANDE: ANALISI DI TIPO E VISITOR

DOMANDA 1: Cosa si fa con l'Analisi di Tipo?

Il controllo di tipo, (type checking) è l'analisi principale della fase di analisi semantica di un linguaggio tipato staticamente. I linguaggi moderni (Java, C#, ecc..) specificano anche altre analisi, ad esempio,l'analisi di raggiungibilità di istruzioni cioè individuare codice che NON verrà mai eseguito, l'analisi di inizializzazione di variabili, ecc.

Per linguaggi molto semplici, l'analisi di tipo si può effettuare durante il parsing, calcolando il tipo delle componenti e mantenendo la symbol table come struttura globale.

Per linguaggi più complessi questa analisi viene fatta visitando (anche più volte) l'AST del programma. Noi effettueremo il controllo di tipi con una visita del AST del programma.

DOMANDA 2: A cosa serve il Pattern Visitor. Descriverne la struttura?

Risposta 1 - Il visitor è un pattern comportamentale usato nella programmazione ad oggetti. Permette di separare un algoritmo dalla struttura di oggetti composti a cui è applicato, in modo da poter aggiungere nuove operazioni e comportamenti senza dover modificare la struttura.

Il funzionamento è il seguente: un client che voglia utilizzare un visitor deve creare un oggetto concreteVisitor e utilizzarlo per attraversare la struttura, chiamando il metodo accept di ogni oggetto. Ogni chiamata invoca nel concreteVisitor il metodo corrispondente alla classe dell'oggettochiamante, che passa sé stesso come parametro per fornire al Visitor un punto d'accesso al propriostato interno.

Risposta 2 - Il pattern Visitor separa la logica di elaborazione di una struttura dati da quella struttura stessa, permettendo di aggiungere nuove operazioni sulla struttura dati senza dover modificare la struttura stessa. La struttura prevede l'utilizzo di una classe Visitor che rappresenta l'operazione da eseguire, una interfaccia Element che rappresenta gli elementi della struttura dati e una classe ConcreteElement che implementa l'interfaccia Element e accetta il Visitor.

DOMANDA

Consideriamo il seguente linguaggio di espressioni intere e floating point:

inum [0-9] +

fnum [0-9]+.[0-9]+

Expr -> Expr plus Expr

| Expr times Expr

| Val

Val -> inum | fnum

Assumiamo che le operazioni plus e times possano avvenire solo se i due operandi sono interi oppure floating point, scrivere:

- Una espressione corretta,
- Una con errori lessicali,
- Una senza errori lessicali ma con errori sintattici
- Ed una con solo errori semantici

ESERCIZIO 1
Data la grammatica:
1: S -> A C $
2: C -> c
3: C -> ε
4: A -> A B C d c
5: A -> B Q
6: B -> b B
7: B -> ε
8: Q -> q
9: Q -> ε

- Trovare i non-terminali e le produzioni che generano la stringa vuota.
- Gli insiemi First delle parti destre delle produzioni e Follow dei non terminali.
- Gli insiemi Predict delle produzioni.

ESERCIZIO 2
Dire se le seguenti grammatiche sono LL(1) oppure no
1: S -> A B c $
2: A -> a
3: A -> ε
4: B -> b
5: B -> ε

ESERCIZIO 3
Dire se le seguenti grammatiche sono LL(1) oppure no
1: S -> A b $
2: A -> a
3: A -> B
4: A -> ε
5: B -> b
6: B -> ε

ESERCIZIO 4
Dire come mai la seguente grammatica non è LL(1) e se possibile definirne una equivalente LL(1)
1: DL -> DL ; D
2: DL -> D
3: D -> T IdL
4: IdL -> IdL , id
5: IdL -> id
4: T -> int 7:
4: T -> bool
Il punto e virgola e la virgola a destra nelle produzioni e id, int e bool sono simboli terminali.

ESERCIZIO 5
Consideriamo la grammatica
Expr -> Expr plus Term | Term
Term -> Term times Val | Val
Val -> inum | fnum
Definire una grammatica per lo stesso linguaggio che permetta il parsing top-down e definire i Predict delle produzioni.