Domanda:Descrivere il ruolo dell'analisi sintattica all'interno del processo di compilazione.
Chiarire perché si fa uso di grammatiche context-free.

Risposta: In questa fase il parser utilizza i token per definire la struttura grammaticale della sequenza di token. Ovvero la struttura delle singole istruzioni e controlla che sono state rispettate le regole sintattiche del linguaggio.
In questa fase i token diventano simboli terminali della grammatica.
L'output di questa fase sarà l'insieme degli alberi sintattici che descrivono il programma.
Il parser genera un albero sintattico per ogni istruzione adoperando il codice semplificato e la symble table costruiti nell'analisi lessicale.
Si fa uso delle grammatiche context-free poiché la classe dei linguaggi CF coincide con quella dei linguaggi accettati da automi a pila

Domanda: Qual'è la differenza tra parser deterministici e non deterministici?
Implementano modelli equivalenti?

Risposta: Il linguaggio riconosciuto da un parser deterministico è context-free e può essere generato da una grammatica non ambigua.
La famiglia dei linguaggi CF deterministici è strettamente contenuta in quella dei linguaggi context-free. In particolare se un linguaggio è inerentemente ambiguo allora il parser non può essere deterministico.
Inoltre esistono linguaggi non ambigui ma il cui parser è effettuato da un parser non deterministico.
Entrambi i parser implementano automi a pila

Domanda:Descrivere la relazione tra la famiglia dei linguaggi da grammatiche LL e quella dei linguaggi generati da grammatiche LR.

Risposta:Ogni linguaggio regolare è LL(1).
Ogni linguaggio LL(k) è deterministico, ma ci sono linguaggi deterministici per cui non esiste una grammatica LL(k). Le grammatiche LL(1) e LR(0) non sono incluse una nell'altra, ma per ogni k>=0 una grammatica LL(k) è anche LR(k);

Domanda:Descrivere l'input e l'output di un analizzatore sintattico e specificare il ruolo delle grammatiche context-free nella fase di analisi sintattica.

Risposta:n questa fase il parser utilizza i token per definire la struttura grammaticale della sequenza di token. Ovvero la struttura delle singole istruzioni e controlla che sono state rispettate le regole sintattiche del linguaggio.
In questa fase i token diventano simboli terminali della grammatica.
L'output di questa fase sarà l'insieme degli alberi sintattici che descrivono il programma.
Il parser genera un albero sintattico per ogni istruzione adoperando il codice semplificato e la symble table costruiti nell'analisi lessicale.
Si fa uso delle grammatiche context-free poiché la classe dei linguaggi CF coincide con quella dei linguaggi accettati da automi a pila

Domanda:Descrivere il funzionamento dei parser LR e la gerarchia delle grammatiche LR, concentrandosi sulla differenza tra grammatiche LR(1) e LALR.

Risposta: sono parser ascendenti, ovvero analizzano una stringa di input cercando di ricostruire i passi di una derivazione rightmost.
Ad ogni passo di riduzione una particale stringa che ha un match con la parte destra di una regola di produzione viene sostituita con la parte sinistra di quella operazione.
Se la sottostringa è scelta in modo corretto allora viene prodotta una derivazione rightmost in ordine inverso
La famiglia dei linguaggi verificabili da parser deterministici coincide con qella dei lingaggi generati da grammatiche LR(1).
La famiglia delle linguaggi generati della grammatiche LR(k) coincide con quelle dei linguaggi generati da LR(1).Per ogni k>=1 esistono grammatiche LR(k) ma non LR(k-1).

Domanda:Durante la fase di analisi sintattica è possibile controllare che una variabile sia stata dichiarata prima di essere utilizzata? Se si, come? Se no, perché?

Risposta:La risposta è no perchè il linguaggio che astrae il problema di verificare che una variabile sia stata dichiarata prima del suo utilizzo, ossia L={ wcw | w,c in sigma*} non è Contex-Free quindi va oltre le potenzialità di un semplice parser.

Teoria:

Perchè usare le grammatiche:
Una grammatica fornisce in modo semplice e facile da capire una specifica della sintassi di un linguaggio di programmazione.
A partire da certe classi di grammatiche è possibile costruire in modo automatico parser efficienti in grado di stabilire se un certo programma è ben formato.
Un parser può rilevare alcune ambiguità sintattiche difficili da notare in fase di progettazione dei linguaggi.
Una grammatica progettata adeguatamente impartisce una struttura ad un linguaggio di programmazione che + utile per la traduzione in codice oggetto e per la ricerca degli errori.

Grammatiche CF e automi a Pila.

La classe dei linguaggi CF coincide con quella degli automi a pila.
Tuttavia il modello da considerare è non deterministico poiché la classe dei linguaggi riconosciuti da automi a pila deterministici è strettamente inclusa in quella dei linguaggi CF. Ad ogni passo l'automa sceglie in modo non deterministico una delle regole applicabili in funzione dello stato, del simbolo corrente e del simbolo in cima alla pila.
Lautoma simula le derivazioni sinistre della grammatica.

In un generico automa a pila sono presenti tre forme di non determinismo.

- Incertezza tra più mosse di lettura
- incertezza tra una mossa spontanea(senza lettura) e una mossa di lettura
- Incertezza fra più mosse spontanee

se nessuna delle tre forte è presente è deterministico e il linguaggio è detto context-free deterministico.
Se un parser è deterministico allora ogni frase è riconosciuta o non riconosciuta con un solo calcolo
Un linguaggio riconosciuto da parser deterministico si dice linguaggio context-free deterministico e può essere generato da una grammatica non ambigua.
La famiglia dei linguaggi context-free deterministici è strettamente inclusa in quella dei linguaggi contex-free.
Se un linguaggio è inerentemente ambiguo allora il parser non può essere mai deterministico.

ALGORITMO DI EARLY:
L'idea è quella di costruire progressivamente tutte le possibili derivazione leftmost o rightmost compatibili con la stringa di input. Durante il processo si analizza la stringa in input da sinistra verso destra scartando via via le derivazioni in cui non vi sia corrispondeza tra i isimboli derivati e quelli della stringa.
Se esistono due derivazioni leftmost o rightmost possibili l'algoritmo restituisce i due alberi di derivazione.
La complessità di calcolo è proporzionale al cubo della lunghezza della stringa da analizzare e si riduce al quadrato se la grammatica è non ambigua e ancora di più se è deterministica.

Complessità di calcolo:
ciascun insieme S[j] può avere un numero di coppie che cresce linearmente con j, quindi O(n);
le operazioni di scansione e predizione su ogni oppia sono indipendenti da n
le operazioni di completamento richiede O(j) per ogni coppia per un totale O(n^2)
sommando i passi si ottiene O(n^3)

STRATEGIE DI RIPARAZIONE:

- panic mode: scoperto l'errore il parser riprende l'analisi in corrispondenza di alcuni token sincronizzanti predefiniti scartando alcuni caratteri. Svantaggi: può essere scordato molto input.
- Phrase level: correzioni locali ottenute inserendo, modificando, cancellando alcuni terminali per poter riprendere l'analisi. Svantaggi: difficoltà quando la distanza dell'errore è notevole
- error productions: uso di produzioni che estendono la grammatica per generare gli errori più comuni. Metono efficiente per la diagnostica
- global correction: si cerca di calcolare la migliore correzione possibile alla derivazione errata

PARSER DISCENDENTI E ASCENDENTI:
discedenti: si costruisce la derivazione partendo dall'assioma,albero di derivazione si costruisce dalla radice alle foglie
ascendenti: si costruisce la derivazione ma nell'ordine riflesso, cioè l'albero si costruisce dalle foglie alla radice.

PARSER A DISCESA RICORSIVA.(parser discendente):
le regole grammaticali per un non-terminale A sono viste come costituenti una procedura che riconosce un A.
la parte destra di ogni regola specifica la struttura del codice di questa procedura
la sequenza di terminali e non terminali nelle varie regole corrisponde a un controllo che i terminali siano presenti nell'input e a invocazioni delle procedure dei simboli non terminali.
In presenza di diverse regole per A è modellate da case e if

PARSER PREDITTIVI:
basandoci sull'input che resta da leggere , predice quale produzione usare senza fare uso del backtraking.
Nella tecnica a discesa ricorsiva l'analisi sintattica viene effettuata attraverso una cascata di chiamate ricorsive.
I parser discendenti deterministici possono anche essere guidati da una tabella. In tale casi si parla di parsr predittivi.
Nel parser LL(1) la pila delle chiamate ricorsive viene esplicitata nel parser, e quindi non si fa uso di ricorsione.

PARSER LL(1):
è un parser predittivo
ha il seguente significato:
la prima L significa che l'input è analizzato da sinistra verso destra
la seconda L significa che il parser costruisce una derivazione leftmost per la stringa di input
il numero 1 significa che l'algoritmo utilizza soltando un solo simbolo dell'input per risolvere le scelte del parser.

Il parser consiste di una pila che continee inizialemnte il simbolo $ (fondo della pila) ed un input la cui fine è marcata con il simbolo $.
Il parser inizia inserendo in simbolo iniziale in testa alla pila.
Il pasers accetta una stringa di inpt se dopo una sequenza di azioni la pila contiene $ e la stringa di input è $.

ogni volta che in testa alla pila c'è un simbolo non terminale X, lo si espande secondo una produzione X->y che viene scelta a seconda del sibolo in testa all''input e ai valori di una tabella.
Ogni volta che sulla pila c'è un simbolo terminale t, si controlla che intesta all'input ci sia lo stesso simbolo, in questo caso lo si elinima sia dalla pila che dall'input, altrimenti è errore.

La complessità di calcolo è lineare nella lunghezza n della stringa sorgente, poiché viene consumanto un

carattere alla volta.

COME OTTENERE GRAMMATICHE LL(1)

- verificare , se è possibile, che non sia ambigua. In caso contrario se si può si rimuova l'ambiguità
- controllare che non ci siano ricorsioni sinistre. In caso contrario trasformarle in ricorsioni destre
- se un simbolo non terminale ammette più derivazioni con lo stesso prefisso applicare il metodo di fattorizzazione sinistra
- in alternativa, può essere necessario allungare la lunghezza della prospezione.

LIMITI DELLA FAMIGLIA LL(k)
non tutti i linguaggi verificabili da parser deterministici sono generabili da grammatiche LL(K)

PARSER ASCENDENTI
gli algoritmi bottom-up analizzano una stringa di input cercando di ricostruire i passi di una derivazione rightmost.
Sono detti bottom-up poiché costruiscono l'albero sintattico relativo ad una stringa prodotta dalla grammatica dalle foglie alla radice.
Un modello generale di parsing bottom-up è il parsing shift-reduce.
La classe più grande di grammatiche per cui è possibile usare un SR parsing è data dalle grammatiche LR.

Nei parsing bottom-up ad ogni passo di riduzione una particolare sottostringa che ha un match con la parte destra di una quale regola di produzione viene sostituita con la paarte sinistra di quella produzione. Se la sottostringa è scelta in modo correstto allora viene così prodotta una derivazione rightmost in ordine inverso..

Hadle: è una sottostringa di una forma sentenziona destra che coincide con la parte destra di una produzione e la cui riduzione rappresenta un passo della inversa della derivazione rightmost.

LR(k):

- L significa che l'input è analizzato da sinistra verso destra
- R significa che il parser produce una derivazione rightmost per la stringa di input
- il numero k significa che l'algoritmo utilizza k simboli dell'input per decidere l'azione del parser.

PERCHè USARE I PARSER LR
possono essere costruiti per riconoscere virtualmente tutti i costrutti di linguaggi di programmazione definiti da grammatiche CF.
riconosce velocemente errori sintattici.
La classe delel grammatiche L contiene propriamente le grammatiche LL. infatti un parser LR(k) deve riconoscere l'occorrenza di una parte destra di produzione in una forma sentenziale destra con k simboli di prospezione. Un parser LL(k) deve scegliere una produzione in base ai primi k simboli della stringa da derivare.

GRAMMATICHE lr(0)
è una grammaica in cui ogni cella della tabella LR(0) contiene al più un valore.
Equivalentemente gli stati dell'automa sono contenteni solo produzioni che presuppongono shift o solo produzione che presuppongono reduce.
Gli stati che presuppongono operazioni di reduce contengono una sola produzione.

Prorpietà dellautoma LR(0):

- tutte le frecce entranti in uno stato hanno la stessa etichetta
- uno stato di reduce non ha successori
- uno stato di shift ha almeno un successore

LR PARSING:

le grammatiche LR sono più potenti delle LL

CONFRONTO TRA GRAMMATICHE E LINGUAGGI LR(0) E LL(1)
le classi di grammatiche LR(0) e LL(1) non sono incluse una nell'altra

- una grammatica che regole vuote non è LR(0) ma può essere LL(1)
- una grammatica che contiene ricorsioni sinistre non è LL(1) ma può essere LR(0)

le famiglie dei linguaggi LR(0) e LL(1) sono distinte e incompatibili

- un linguaggio chiso per prefissi non è LR(0) ma può essere LL(1)
- esistono linguaggi LR(0) ma non LL(1)

PROPRIETà DEI LINGUAGGI E DELLE GRAMMATICHE lr(K)

la famiglia dei linguaggi verificabili da parser determinstici coincide con quella dei linguaggi generati dalle grammatiche LR(1). Ciò non significa che ogni grammatica il cui linguaggio è deterministo sia necessariamente LR(1).
La famiglia dei linguaggi generati dalle grammatiche LR(k= coincide con quella dei linguaggi generati da LR(1). Quindi un linguaggio context-free ma non-deterministico non può avere una grammatica LR(k)
per ogni k>=1 esistono grammatiche LR(k) ma non LR(k-1)
data una grammatica è indecidibile se esista un k>0 per cui tale grammatica risulti LR(k), di conseguenza non è decidibile se il linguaggio generato da una grammatica CF è deterministo. È indecidibile soltanto se k è fissato.