DOMANDA 1: Consideriamo un linguaggio, L, come si specificano il lessico di L e la sintassi di L?
Il lessico è l'insieme delle parole e dei simboli appartenenti ad un linguaggio. La stringa di caratteri rappresenta un particolare token.
La sintassi di un linguaggio è l'insieme di regole che una porzione di codice deve seguire per essere considerata conforme a quel linguaggio.

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 finiti deterministici.
Esempio analisi lessicale
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 è eseguita dal parser e in questa fase si riconosce se un token è derivabile e si trovano errori sintattici. 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:
1)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.
2)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 eseguita dal **type checker** che visita l'ast e controlla:
1)Che le variabili siano state dichiarate prima di essere usate e siano dichiarate una volta sola,
2)Che siano stati usati i giusti tipi. All'occorrenza si effettua una conversione,
3)La raggiungibilità del codice,
4)Corretto ritorno di valori dai metodi.
In questa fase, nel caso di approccio alla traduzione in due fasi, il type checker aggiunge nell'ast delle informazioni di tipo.
L'input di questa fase ho come input l'ast mentre come output ho di nuovo l'ast ma ho controllato se il modo in cui sono organizzati i nodi è corretto.

→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?

DOMANDA 5: Come si può implementare il riconoscimento lessicale?

→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 una produzione. 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 $1 \leq i \neq j \leq n$.

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 $\rightarrow$ E + T | E − T | v

Perché? Assumi A $\rightarrow$ 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 ⇒ A α ⇒ α

Una grammatica con prefissi comuni, cioè 2 o più produzioni per lo stesso non terminale hanno la stessa parte iniziale, non può essere LL(1).

Ad esempio, S $\rightarrow$ if E then E | if E then E else E.

Perché?

→se A $\rightarrow$ α β 1 | α β 2, se a ∈ First(α β 1) allora a ∈ First(α β 2), 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?

Esistono due tecniche di parsing:

1)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.

2)BOTTOM-UP: L'ast si costruisce a partire dalla grammatica LR e si costruisce l'albero partendo dalle foglie e proseguendo verso la radice.

→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 (AST) è una rappresentazione ad albero del codice sorgente. Ogni nodo dell'albero è un costrutto che si verifica nel codice sorgente. Un AST è il risultato della fase di analisi sintattica del compilatore.

DOMANDA 2: Quale è la differenza fra AST e parsing tree per una stringa di un 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 ed ogni volta che un identificatore viene usato, la tabella di simboli consente di accedere alle informazioni raccolte quando è stato definito. In genere, si implementa tramite una tabella hash.

→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?

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'oggetto chiamante, che passa sé stesso come parametro per fornire al Visitor un punto d'accesso al proprio stato interno.

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

ESERCIZI (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

DOMANDA 1: Trovare i non-terminali e le produzioni che generano la stringa vuota.

DOMANDA 2: Gli insiemi First delle parti destre delle produzioni e Follow dei non terminali.

DOMANDA 3: Gli insiemi Predict delle produzioni.

ESERCIZI (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

ESERCIZI (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

Esercizi: (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

6: T → int

7: T → bool

Il punto e virgola e la virgola a destra nelle produzioni e id, int e bool sono simboli terminali.

Esercizi: (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.