

A decorative graphic on the left side of the slide consists of a grid of colored squares. The squares are arranged in a pattern that roughly forms a staircase or a series of overlapping steps. The colors include yellow, light teal, and dark teal. The squares are of varying sizes and are positioned to the left of the main text area.

Linguaggi di Programmazione (Corso A)

Docente: Giovanni Semeraro

Il Modello di un Compilatore



Il Modello di un Compilatore

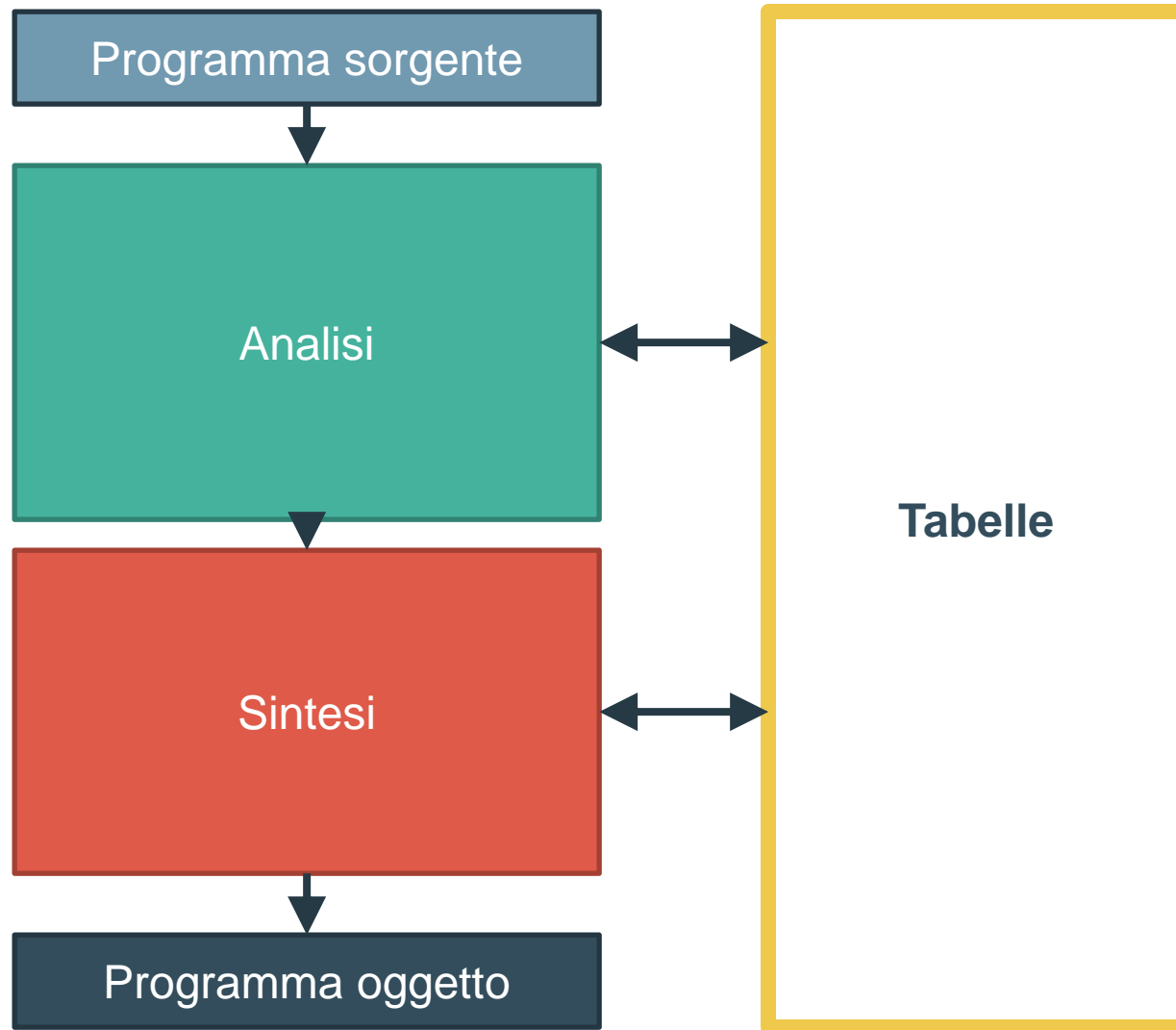
- La costruzione di un compilatore per un particolare linguaggio di programmazione è un'operazione abbastanza complessa.
- La complessità dipende principalmente dal linguaggio sorgente.



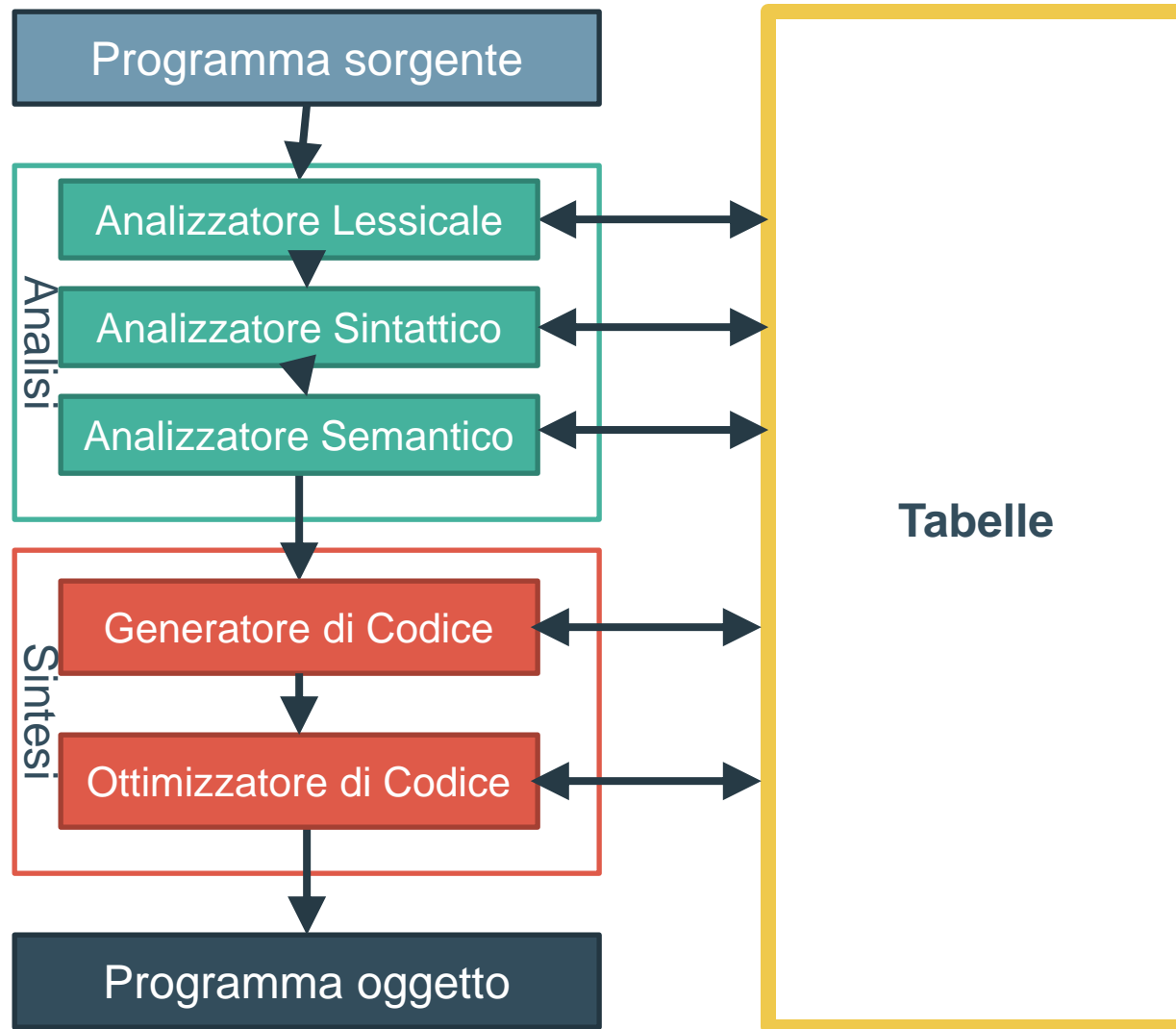
Compilatore

- Traduce il programma sorgente in programma oggetto.
- Esegue:
 - **Analisi** del programma sorgente;
 - **Sintesi** del programma oggetto.

Compilatore: modello funzionale



Compilatore: modello funzionale





Programma sorgente

- È una stringa di simboli.
- **Esempio**

```
if A>B then X:=Y;
```



Analizzatore lessicale (scanner)

- **Input:** un programma sorgente
- Esamina il programma per individuare i simboli (token) che lo compongono classificando parole chiave, identificatori, operatori, costanti, ecc.
- Per ragioni di efficienza ad ogni classe di token è dato un numero unico che la identifica.
- **Output:** lista di token

Analizzatore lessicale (scanner)

■ Esempio

```
if A>B then X:=Y;
```

IF	20
A	1
>	15
B	1
THEN	20
X	1
:=	10
Y	1
;	27

- Si noti che vengono ignorati spazi bianchi e commenti. Inoltre alcuni scanner inseriscono label, costanti e variabili in tavole appropriate.
- Un elemento della tavola per una variabile, ad esempio, contiene nome, tipo, indirizzo, valore e linea in cui è dichiarata.

Analizzatore lessicale (scanner)

■ Esempio

Programma in input

```
x1 := a + bb * 12 ;  
x2 := a / 2 + bb * 12 ;
```

Sequenza di token

"x1"	Id
":="	Op
"a"	Id
"+"	Op
"bb"	Id
"*"	Op
12	Lit
;	Punct
"x2"	Id
":="	Op
"a"	Id
"/"	Op
2	Lit
"+"	Op
"bb"	Id
"*"	Op
12	Lit
;	Punct

Analizzatore lessicale (scanner)

■ Esempio

Programma in input

```
x1 := a + bb * 12 ;  
x2 := a / 2 + bb * 12 ;
```

Sequenza di token

"x1"	Id
":="	Op
"a"	Id
"+"	Op
"bb"	Id
"*"	Op
12	Lit
;	Punct
"x2"	Id
":="	Op
"a"	Id
"/"	Op
2	Lit
"+"	Op
"bb"	Id
"*"	Op
12	Lit
;	Punct

Analizzatore lessicale (scanner)

■ Esempio

Programma in input

```
x1 := a + bb * 12 ;  
x2 := a / 2 + bb * 12 ;
```

Sequenza di token

"x1"	Id
":="	Op
"a"	Id
"+"	Op
"bb"	Id
"*"	Op
12	Lit
;	Punct
"x2"	Id
":="	Op
"a"	Id
"/"	Op
2	Lit
"+"	Op
"bb"	Id
"*"	Op
12	Lit
;	Punct

Analizzatore lessicale (scanner)

■ Esempio

Programma in input

```
x1 := a + bb * 12 ;  
x2 := a / 2 + bb * 12 ;
```

Sequenza di token

"x1"	Id
":="	Op
"a"	Id
"+"	Op
"bb"	Id
"*"	Op
12	Lit
;	Punct
"x2"	Id
":="	Op
"a"	Id
"/"	Op
2	Lit
"+"	Op
"bb"	Id
"*"	Op
12	Lit
;	Punct

Analizzatore lessicale (scanner)

■ Esempio

Programma in input

```
x1 := a + bb * 12 ;  
x2 := a / 2 + bb * 12 ;
```

Sequenza di token

"x1"	Id
":="	Op
"a"	Id
"+"	Op
"bb"	Id
"*"	Op
12	Lit
;	Punct
"x2"	Id
":="	Op
"a"	Id
"/"	Op
2	Lit
"+"	Op
"bb"	Id
"*"	Op
12	Lit
;	Punct



Analizzatore sintattico (parser)

- **Input:** lista di token
- Individua la struttura sintattica della stringa in esame a partire dal programma sorgente sotto forma di token.
- Identifica quindi espressioni, istruzioni, procedure.
- **Output:** albero sintattico

Analizzatore sintattico (parser)

- **Esempio** `ALFA1 := 5 + A * B`
- La stringa `5 + A * B` è riconosciuta come `<espressione>`
- La stringa completa è riconosciuta come `<assegnazione>` in accordo alla regola sintattica:

`<assegnazione> ::= <variabile> := <espressione>`

- In realtà si utilizza la stringa semplificata del tipo:

`id1 := c2 + id3 * id4`

con accesso alla rappresentazione generata dallo scanner.

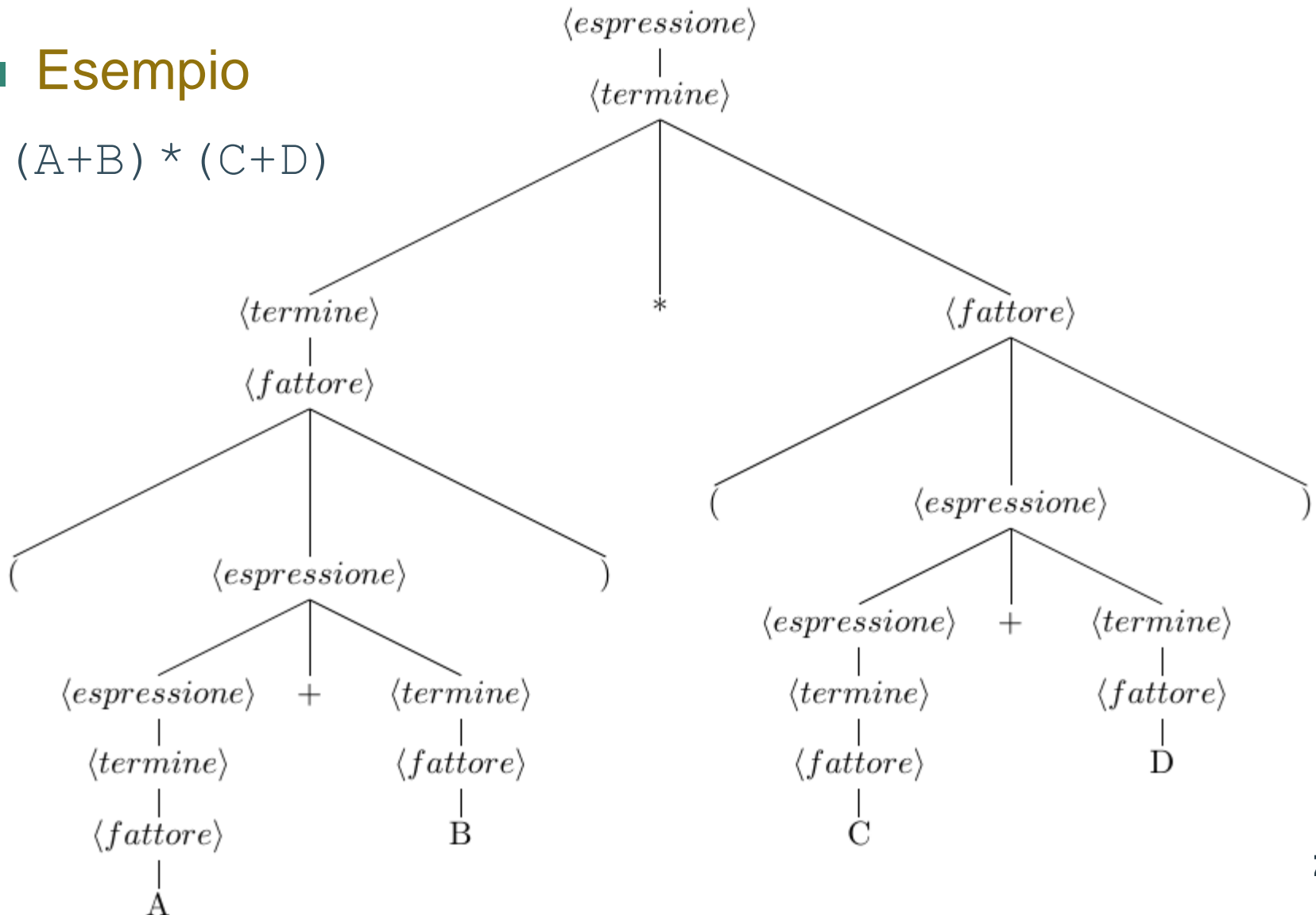
Analizzatore sintattico (parser)

- **Esempio** $(A+B) * (C+D)$
- L'analisi produce le classi sintattiche:
 - `<fattore>`
 - `<termine>`
 - `<espressione>`
- Il controllo sintattico si basa sulle *regole grammaticali* utilizzate per definire formalmente il linguaggio.
- Durante il controllo sintattico si genera l'albero di derivazione (*albero sintattico*)

Analizzatore sintattico (parser)

■ Esempio

$(A+B) * (C+D)$

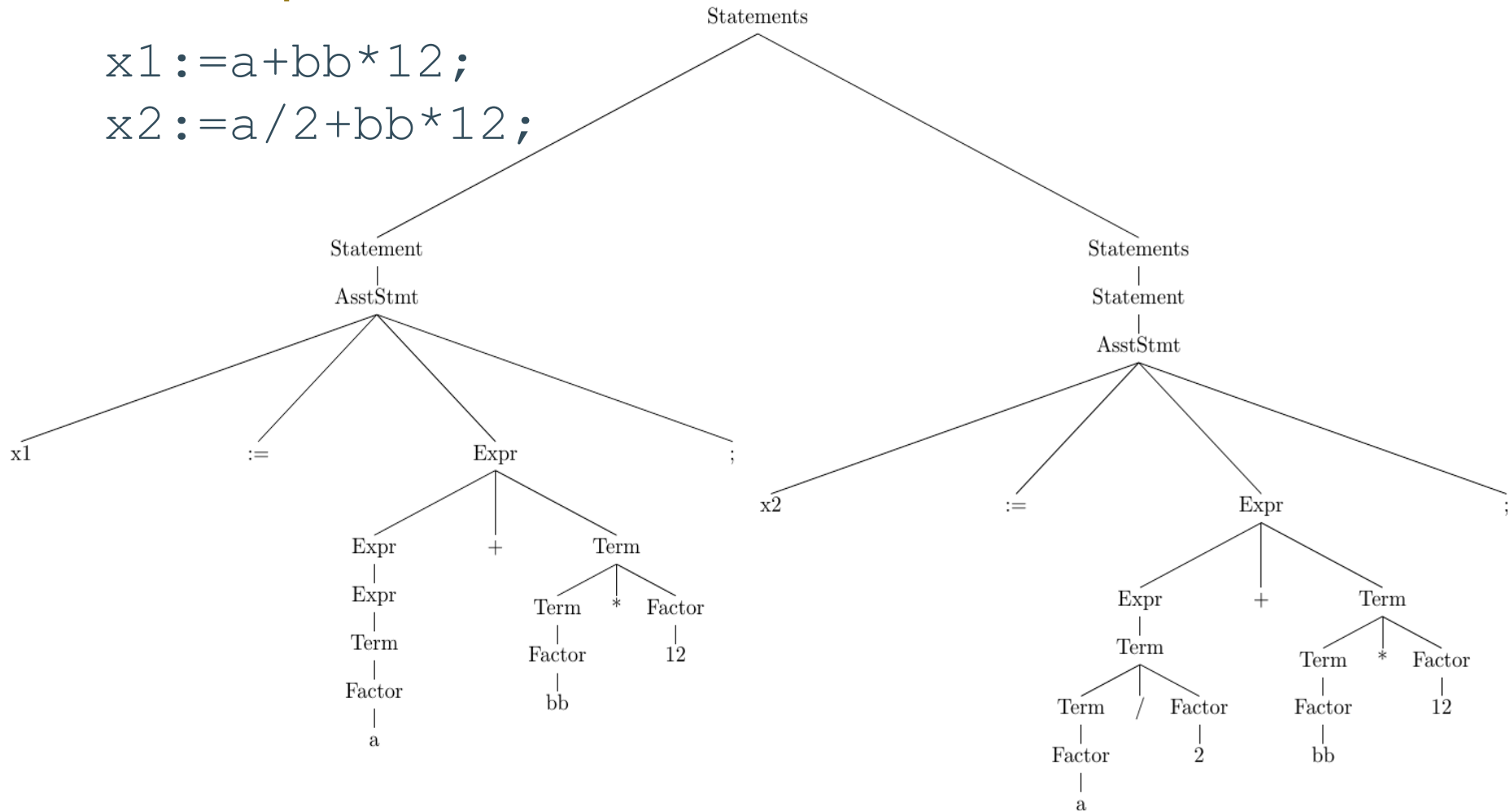


Analizzatore sintattico (parser)

■ Esempio

`x1 := a + bb * 12 ;`

`x2 := a / 2 + bb * 12 ;`





Analizzatore semantico

- **Input:** albero sintattico generato dal parser.
- Si compone di due fasi principali:
 1. **Controlli statici** (static checking).
 2. **Generazione di una rappresentazione intermedia (IR)**
- **Output:** albero arricchito con informazioni sui vincoli sintattici contestuali



Analizzatore semantico

- Input: albero sintattico generato dal parser.
- Si compone di due fasi principali:
 1. **Controlli statici** (static checking).
Sono svolti vari controlli sui tipi, dichiarazioni, numero parametri funzioni, etc.

Per l'espressione $(A+B) * (C+D)$, ad esempio, l'analizzatore semantico deve determinare quali azioni sono specificate dagli operatori aritmetici di addizione e moltiplicazione.

Ad ogni token che corrisponde ad un identificatore di variabile è associato: tipo, luogo di dichiarazione, etc memorizzate nella tabella dei simboli.

Quando riconosce il $+$ od il $*$ invoca allora una routine semantica che specifica le azioni da svolgere. Ad esempio, che gli operandi siano stati dichiarati, abbiano lo stesso tipo ed un valore.

Analizzatore semantico

- Input: albero sintattico generato dal parser.
- Si compone di due fasi principali:
 2. **Generazione di una rappresentazione intermedia (IR)**
Spesso la parte di analisi semantica produce anche una forma intermedia di codice sorgente.

Ad esempio può produrre il seguente insieme di quadruple per $(A+B) * (C+D)$:

$(+, A, B, T1)$
 $(+, C, D, T2)$
 $(*, T1, T2, T3)$

Od altri tipi di codice intermedio.



Analizzatore semantico

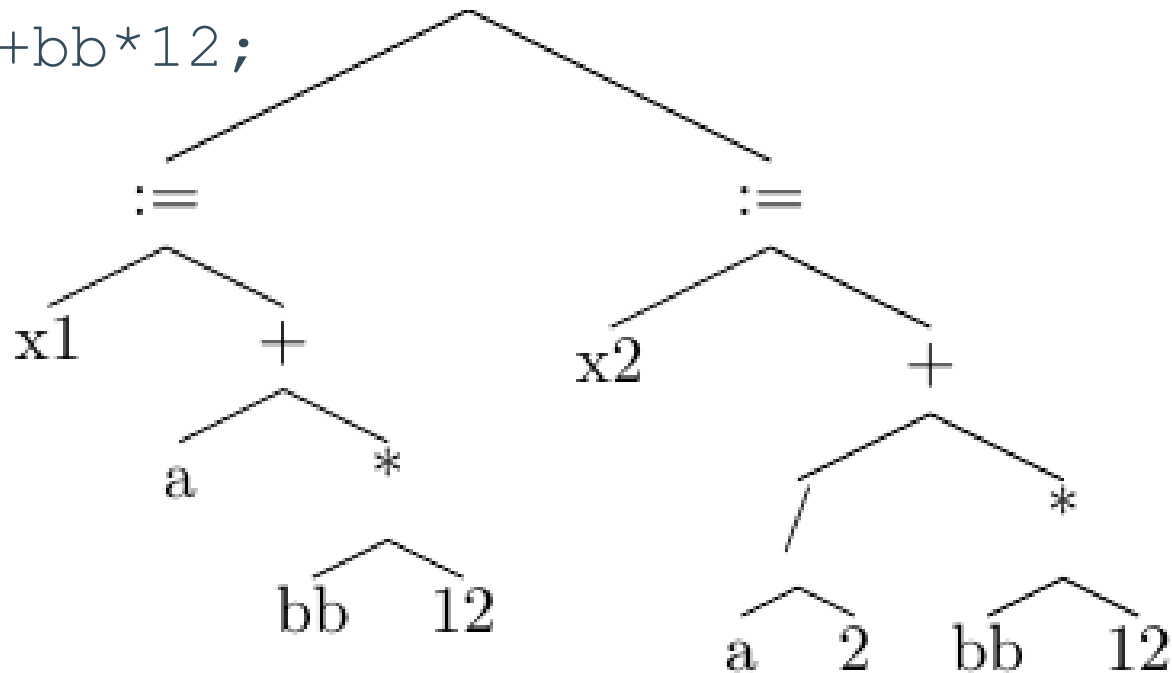
- **Esempio:** codice intermedio che rimuove dall'albero sintattico alcune delle categorie intermedie e mantiene solo la struttura essenziale (**albero sintattico astratto**).
 - Tutti i nodi sono token.
 - Le foglie sono operandi.
 - I nodi intermedi sono operatori.
- Spesso a valle dell'analizzatore semantico ci può essere un ottimizzatore del codice intermedio.

Analizzatore semantico

■ Esempio:

`x1 := a + bb * 12 ;`

`x2 := a / 2 + bb * 12 ;`





Analizzatore semantico

- Ottimizzazione del codice intermedio: **propagazione di costanti**

- **Esempio**

```
X := 3;  
A := B + X;
```

- Si può ottimizzare come:

```
X := 3;  
A := B + 3;
```

evitando un accesso alla memoria.

Analizzatore semantico

- Ottimizzazione del codice intermedio: **eliminazione di sotto-espressioni comuni**

- **Esempio**

$A := B * C;$

$D := B * C;$

- Si trasforma in:

$T := B * C;$

$A := T;$

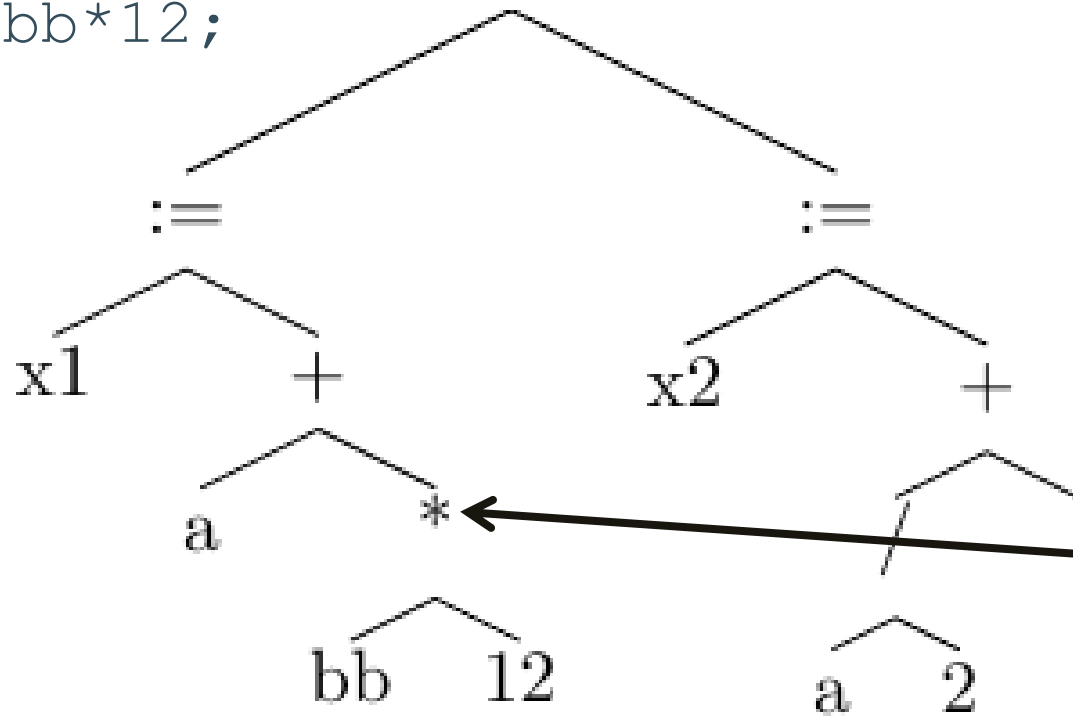
$D := T;$

Analizzatore semantico

■ Esempio

`x1 := a + bb * 12 ;`

`x2 := a / 2 + bb * 12 ;`



Analisi

- **Verifica della correttezza lessicale, sintattica e semantica di un programma**
 - Svolta in fase di compilazione;
 - Verifica che:
 - I simboli utilizzati siano legali, cioè appartengano all'alfabeto (analisi lessicale);
 - Le regole grammaticali siano rispettate (analisi sintattica);
 - I vincoli imposti dal contesto siano rispettati (analisi semantica).
- **Esempio:**

```
var A: integer;
```

```
...
```

```
if A then ...
```

Errore!



Generatore di codice

- L'output dell'analizzatore semantico è passato al generatore di codice che trasla la forma intermedia in linguaggio assembler o macchina;
- Prima della generazione del codice oggetto ci sono delle fasi di preparazione:
 - Allocazione della memoria: può essere allocata staticamente oppure è uno stack o heap la cui dimensione cambia durante l'esecuzione;
 - Allocazione dei registri: poiché l'accesso ai registri è più rapido dell'accesso alle locazioni di memoria, i valori cui si accede più spesso andrebbero mantenuti nei registri.

Generatore di codice

■ Esempio

$x1 := a + bb * 12;$

$x2 := a / 2 + bb * 12;$

- Potremmo pensare di allocare l'espressione $bb * 12$ al registro 1, ed una copia del valore di a al registro 2 assieme al valore $a / 2$.
- Le variabili si potrebbero allocare sullo stack con a al top, e poi, nell'ordine, bb , $x1$, $x2$. Il registro S punta al top dello stack.
- Segue poi la vera e propria generazione di codice.



Generatore di codice

■ Esempio

(+, A, B, T1)
(+, C, D, T2)
(* , T1, T2, T3)

■ Si possono produrre quindi le seguenti istruzioni assembler

```
LOADA  A
LOADB  B
STOREA T1
LOADA  C
LOADB  D
STOREA T2
LOADA  T1
LOADB  T2
MULT
STOREA T3
```

Generatore di codice

- **Esempio** $x1 := a + bb * 12;$
 $x2 := a / 2 + bb * 12;$
- Possiamo generare per una macchina di nostra invenzione il seguente codice
- Nota
 - (S) , $1(S)$, $2(S)$, etc: accede al contenuto del top dello stack, ad una posizione successiva, due posizioni successive, etc.
 - $@A$: accede alla locazione il cui valore è puntato da A (indirizzamento indiretto).

Generatore di codice

- **Esempio** $x1 := a + bb * 12;$
 $x2 := a / 2 + bb * 12;$
- Possiamo generare per una macchina di nostra invenzione il seguente codice

PushAddr x2	Mette l'indirizzo di X2 nello stack
PushAddr x1	Mette l'indirizzo di X1 nello stack
Push bb	Mette bb nello stack
Push a	Mette a nello stack
Load 1(S), R1	Mette bb in R1
Mpy #12, R1	Mette $bb * 12$ in R1
Load (S), R2	Mette a in R2
Store R2, R3	Copia a in R3
Add R1, R3	Mette $a + bb * 12$ in R3
Store R3, @2(S)	Mette $a + bb * 12$ in X1
Div #2, R2	Mette $a / 2$ in R2
Add R1, R2	Mette $a / 2 + bb * 12$ in R2
Store R2, @3(S)	Mette $a / 2 + bb * 12$ in X2



Ottimizzatore di codice

- L'output del generatore di codice è passato in input all'ottimizzatore di codice, presente nei compilatori più sofisticati.
- **Ottimizzazioni indipendenti dalla macchina:** ad esempio la rimozione di istruzioni invarianti all'interno di un loop, fuori dal loop, etc.
- **Ottimizzazioni dipendenti dalla macchina:** ad esempio ottimizzazione dell'uso dei registri

Ottimizzatore di codice

- L'output del generatore di codice è passato in input all'ottimizzatore di codice, presente nei compilatori più sofisticati.
- **Esempio:** ottimizzazione del codice precedente

LOADA A

LOADB B

STOREA T1

LOADA C

LOADB D

STOREA T2

LOADA T1

LOADB T2

MULT

STOREA T3

LOADA A

LOADB B

STOREA T1

LOADA C

LOADB D

LOADB T1

MULT

STOREA T3



Passi di un compilatore

- Scanner e parser possono essere eseguiti in sequenza uno dopo l'altro, producendo prima tutti i token e poi l'analisi sintattica, oppure lo scanner è chiamato dal parser ogni volta che necessita un nuovo token.
- Nel primo caso lo scanner ha esaminato l'intero programma sorgente prima di passare il controllo al parser e quindi ha compiuto un intero passo separato.
- A volte il parser, l'analizzatore semantico ed il generatore di codice sono combinati in un singolo passo. Alcuni compilatori sono solo ad un passo, altri anche fino a 30!



Passi di un compilatore

- Abbiamo ignorato altri aspetti importanti della compilazione:
 1. Error Detection e Recovery;
 2. Le Tabelle dei Simboli prodotte dai vari moduli;
 3. La Gestione della Memoria implicata da alcuni costrutti del linguaggio di alto livello.
- Le fasi di più semplice progettazione, con un apparato formale ben sviluppato e quindi facilmente automatizzabili sono scanner e parser, mentre maggiore difficoltà si trova nella progettazione di analizzatori semantici, generatori ed ottimizzatori di codice.



Linking e Caricamento

- Il programma oggetto prodotto dal compilatore contiene una serie di riferimenti esterni (es. riferimenti a programmi di libreria, funzioni).
- I riferimenti esterni vengono risolti dal **linker**.
- Il programma è rilocabile: può essere allocato in diverse zone di memoria cambiando indirizzo `ind` (indirizzamento relativo).
- Fase di caricamento compiuta dal **loader** che assegna un valore numerico all'indirizzo `ind`, trasformando gli indirizzi relativi in assoluti.

Linking e Caricamento

■ Esempio



Il linker riceve in ingresso questi tre moduli e genera un unico modulo con riferimento ad indirizzi contigui a partire da un indirizzo simbolico `ind`.

Ogni riferimento a moduli esterni viene sostituito con l'indirizzo così calcolato.

Linking e Caricamento

■ Esempio

Indirizzo	Contenuto	Commento
ind	Inizio Padre	
	...	
	salta ad ind + 301	ref. a Figlio1
	...	
	salta ad ind + 421	ref. a Figlio2
	...	
ind + 300	fine Padre	
ind + 301	inizio Figlio1	
	...	
ind + 420	fine Figlio1	
ind + 421	inizio Figlio2	
	...	
ind + 570	fine Figlio2	