# Linguaggi di Programmazione Corso C

## Parte n.1
## Organizzazione del Corso e Introduzione Generale

Nicola Fanizzi (*fanizzi@di.uniba.it*)

**Dipartimento di Informatica**
**Università degli Studi di Bari**

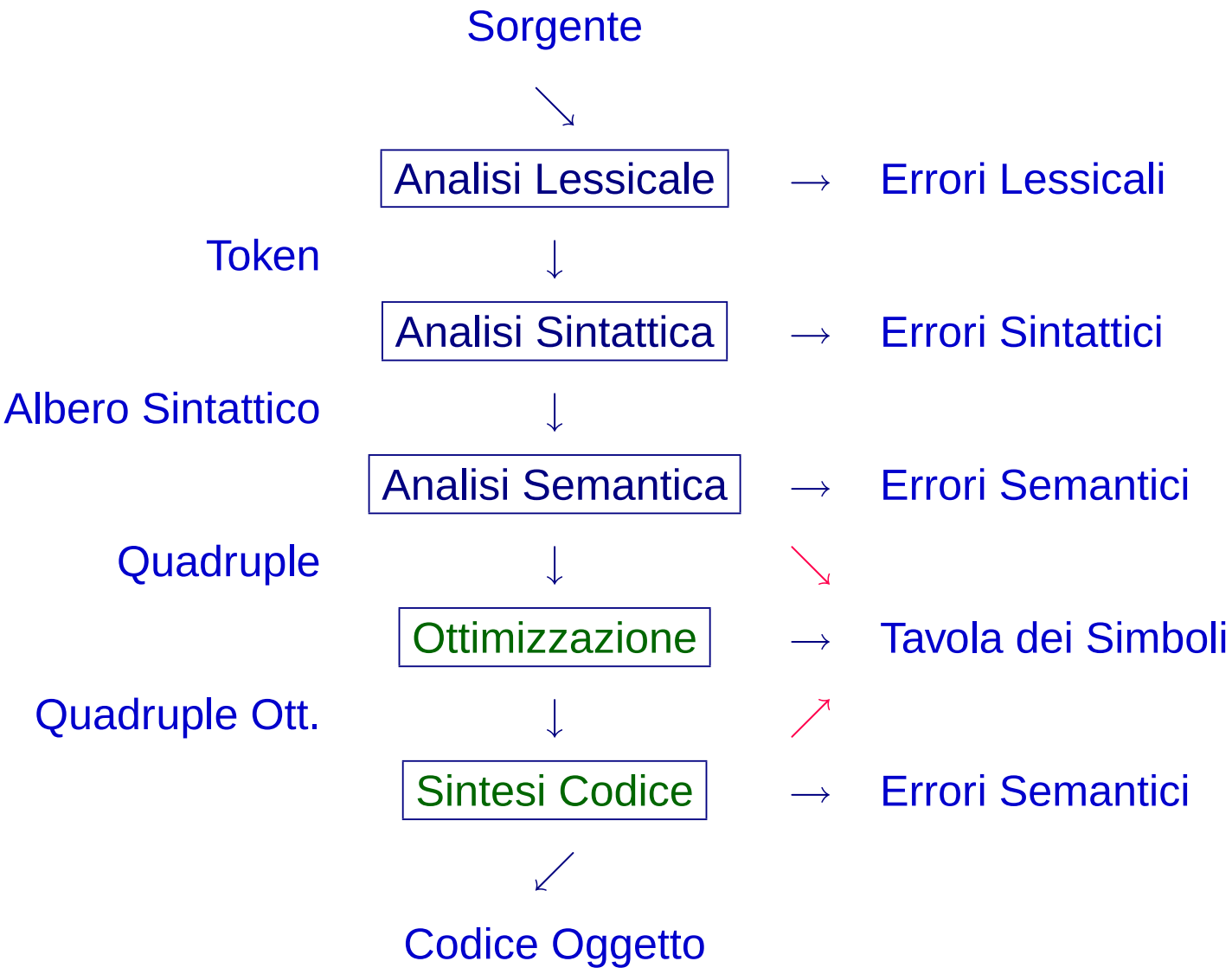
# Organizzazione del Corso

- Sito: **http://lacam.di.uniba.it:8000/∼nico/lingpro/**
- Ricevimento Martedi (11–13) stanza 525, $5^o$ piano DIB
- Prerequisiti: Programmazione, Matematica Discreta
- Obiettivi del corso:
  - Analisi: Teoria dei Linguaggi Formali
  - Sintesi: Compilatori e Produzione del codice
- Laboratorio: Il linguaggio C e la realizzazione di moduli e strutture dati relative ai compilatori
- IDE Dev-C++ sul sito: **http://www.bloodshed.net**
- testi:
  - G. Semeraro - *Appunti di Teoria dei Linguaggi Formali*, Adriatica Ed.
  - G. Bruno - *Linguaggi Formali e Compilatori*, UTET
  - altri, cf. Programma Preliminare

# Panoramica

- **Programmi** Sorgenti
  scritti nei Linguaggi di Programmazione di alto livello (Pascal, C, Basic)
  - potenti
  - rigorosi
  - versatili
- tramite i **Compilatori** ...
- **Codice Oggetto**
  scritto in Linguaggio Macchina
  - comprensibile ai computer
  - efficiente
  - dipendente dalla particolare architettura

# Compilazione




Legenda: **Analisi** e **Sintesi** → denota la produzione ↘ denota l'uso

# Analisi Lessicale

```
int A,B,V,X,Y;
X = Y+A*B*2;
V = X+A*B;
```

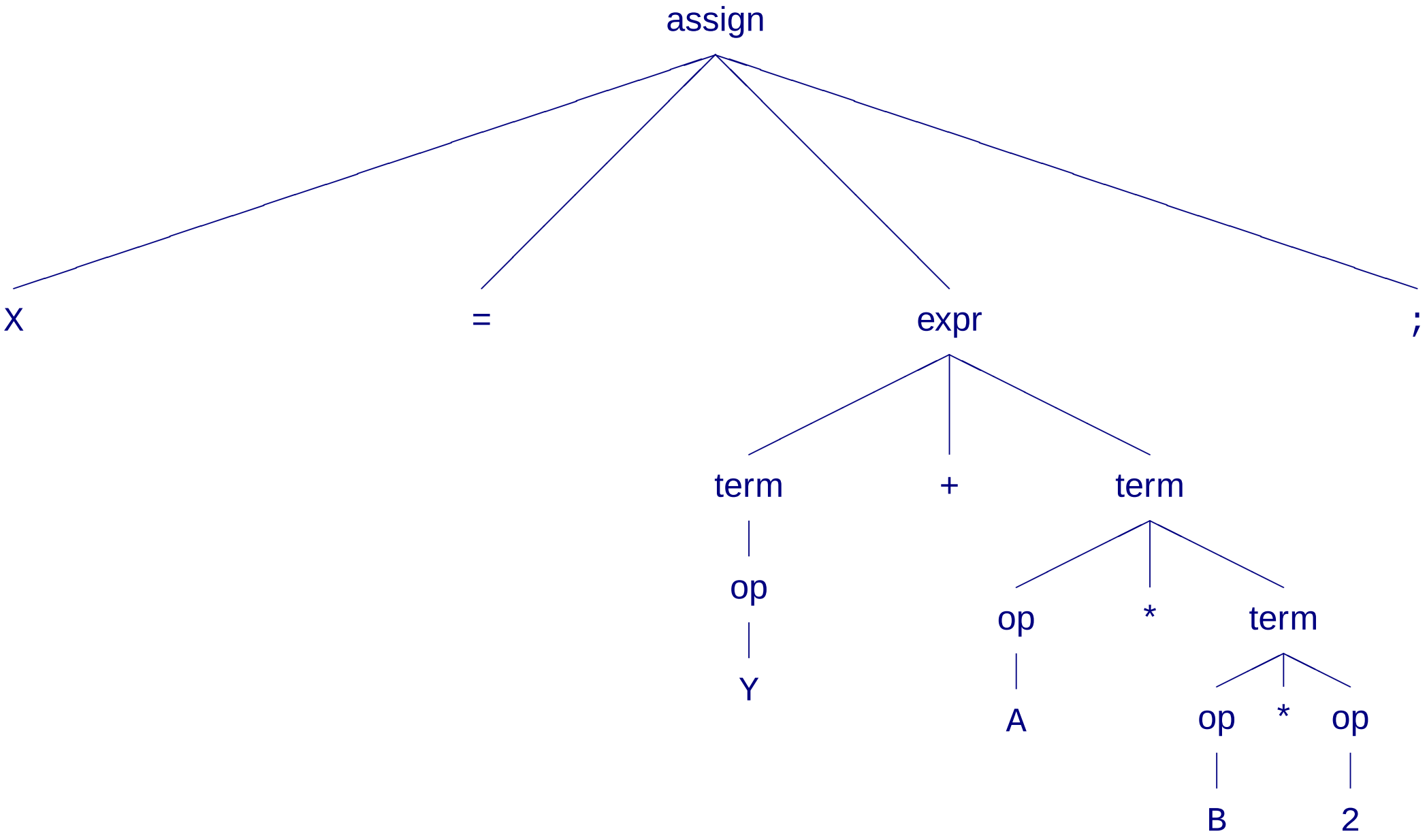
- parole chiave: es. `int`
- identificatori: `A B V X Y`
- costante intera: `2`
- operatori: `= + *`
- separatori: `, ;`

**Flusso di caratteri** convertito in **token**

# Analisi Sintattica

**tokens** convertiti in una struttura di programma **Albero sintattico**: `token` (foglie) e categorie sintattiche (nodi interni)

```
X = Y+A*B*2;
```



# Analisi Semantica

Estrazione del significato del programma:

- **verifica** delle regole del linguaggio, as esempio:
  - dichiarazione effettiva degli identificatori
  - controllo sulla comparabilità dei tipi nelle espressioni
- **costruzione** della tabella dei simboli
  contenente le informazioni sugli identificatori definiti dall'utente;
- **generazione** del codice intermedio comune a più linguaggi (quadruple)

# Esempio

```
X = Y+A*B*2;    V = X+A*B;
```

| nome | tipo | offset |
|---|---|---|
| A | int | 0 |
| B | int | 4 |
| V | int | 8 |
| X | int | 12 |
| Y | int | 16 |
| ... | ... | ... |

quadruple `(istr,op1,op2,dest)`

```
(*,A,B,T1)
(*,T1,2,T2)
(+,Y,T2,T3)
(=,T3,_,X)
(*,A,B,T4)
(+,X,T4,V)
```

# Ottimizzazione

Rendere più efficiente il codice nelle quadruple

```
X = Y+A*B*2;     V = X+A*B;
```

| quadruple |
| --- |
| (*,A,B,T1) |
| (*,T1,2,T2) |
| (+,Y,T2,T3) |
| (+,T1,T3,V) |

# Generazione del Codice

Quindi si genera il codice a seconda del processore

| codice Motorola 68000 |
| --- |
| `MOVE A,D0` |
| `MULS B,D0` |
| `ALS #1,D0` |
| `MOVE Y,D1` |
| `ADD D1,D0` |
| `MOVE D0,V` |