REALIZZAZIONE

- Per la realizzazione di un linguaggio di programmazione e' necessaria la presenza di una macchina (fisica o astratta) che sia in grado di eseguire i programmi del linguaggio.

- Realizzazione di un "traduttore" che renda i programmi eseguibili su un dato elaboratore (compilatore o interprete).

ASTRAZIONE

- Esistono linguaggi a vari livelli di astrazione
  - **Linguaggio Macchina:**
    - implica la conoscenza dei metodi di rappresentazione delle informazioni utilizzati.
  - **Linguaggio Macchina e Assembler:**
    - implica la conoscenza dettagliata delle caratteristiche della macchina (registri, dimensioni dati, set di istruzioni)
    - semplici algoritmi implicano la specifica di molte istruzioni
  - **Linguaggi di Alto Livello:**
    - Il programmatore può astrarre dai dettagli legati all'architettura ed esprimere i propri algoritmi in modo simbolico.

Sono indipendenti dalla macchina hardware sottostante
ASTRAZIONE

ASTRAZIONE

- **Linguaggio Macchina:**

```
0100 0000 0000 1000
0100 0000 0000 1001
0000 0000 0000 1000
```

Difficile leggere e capire un programma scritto in forma binaria

- **Linguaggio Assembler:**

```
… LOADA H
  LOADB Z
  ADD
  …
```

Le istruzioni corrispondono univocamente a quelle macchina, ma vengono espresse tramite nomi simbolici (parole chiave).

- **Linguaggi di Alto Livello:**

```
main()
{ int A;
  scanf("%d",&A);
  if (A==0) {…}
…}
```

Sono indipendenti dalla macchina

ESECUZIONE

- Per eseguire sulla macchina hardware un programma scritto in un linguaggio di alto livello e' necessario tradurre il programma in sequenze di istruzioni di basso livello, direttamente eseguite dal processore, attraverso:

  - interpretazione (ad es. BASIC)
  - compilazione (ad es. C, FORTRAN, Pascal)

COME SVILUPPARE UN PROGRAMMA

- Qualunque sia il linguaggio di programmazione scelto occorre:
  - *Scrivere il* **testo del programma** *e memorizzarlo su supporti di memoria permanenti (fase di editing);*

- Se il linguaggio è compilato:
  - *Compilare il programma, ossia utilizzare il compilatore che effettua una traduzione automatica del programma scritto in un linguaggio qualunque in un programma equivalente scritto in* **linguaggio macchina**;
  - *Eseguire il programma tradotto.*

- Se il linguaggio è interpretato:
  - *Usare l'interprete per eseguire il programma.*

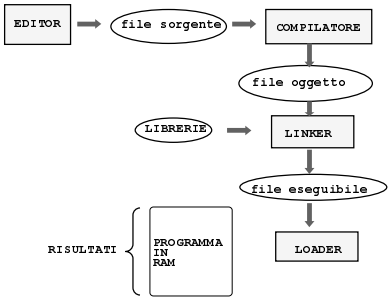
COMPILATORI E INTERPRETI

- I **compilatori** traducono automaticamente un programma dal linguaggio L a quello macchina (per un determinato elaboratore).

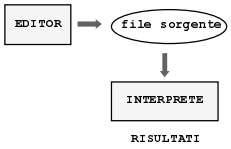
- Gli **interpreti** sono programmi capaci di eseguire direttamente un programma in linguaggio L istruzione per istruzione.

- I programmi compilati sono in generale più efficienti di quelli interpretati.

APPROCCIO COMPILATO: SCHEMA



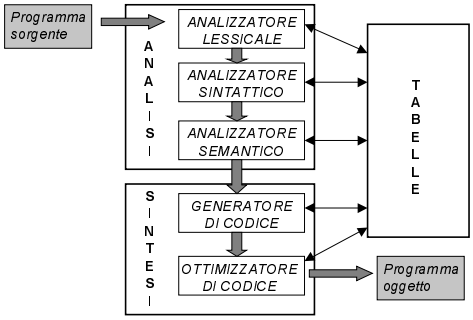
APPROCCIO INTERPRETATO: SCHEMA



COMPILATORI: MODELLO

- La costruzione di un compilatore per un particolare linguaggio di programmazione è complessa.
  - La complessità dipende dal linguaggio sorgente
- Compilatore: traduce il programma sorgente in programma oggetto.
- Due compiti:
  - ANALISI del programma sorgente
  - SINTESI del programma oggetto

COMPILATORI: MODELLO



ANALIZZATORE LESSICALE

- Un programma sorgente è una stringa di simboli
- Analizzatore lessicale o *scanner*: esamina il programma sorgente per identificare i simboli che lo compongono (*tokens*) classificando parole chiave, identificatori, operatori, costanti…..
- Ad ogni classe di tokens è associato un numero unico che la identifica.
- Vengono ignorati spazi bianchi e commenti

ANALIZZATORE LESSICALE

- Esempio: `X1= a+bb*12;`
  `X2= a/2+bb*12;`

TOKEN	CLASSI
X1	id
=	op
a	id
+	op
bb	id
*	op
12	lit
;	punct

TOKEN	CLASSI
X2	id
=	op
a	id
/	op
2	lit
+	op
bb	id
*	op
12	lit
;	punct

2

ANALIZZATORE SINTATTICO

- Analizzatore sintattico o *parser*: individua la struttura sintattica della stringa in esame a partire dal programma sorgente già trasformato sotto forma di tokens: identifica espressioni, istruzioni, procedure…
- Esempio `5 + A*B`
  - la sottostringa 5 + A * B viene riconosciuta come `<espressione>`

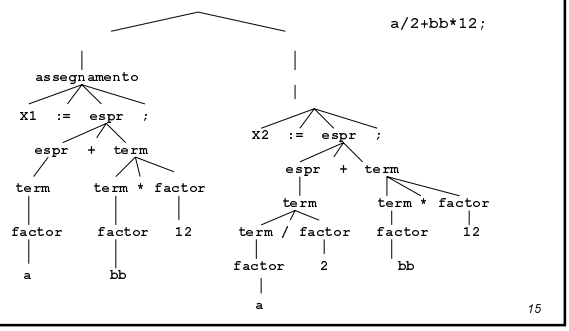
13

ANALIZZATORE SINTATTICO

- Il controllo sintattico si basa sulle regole grammaticali utilizzate per definire formalmente il linguaggio
- Durante il controllo sintattico si genera l'albero di derivazione o *albero sintattico*

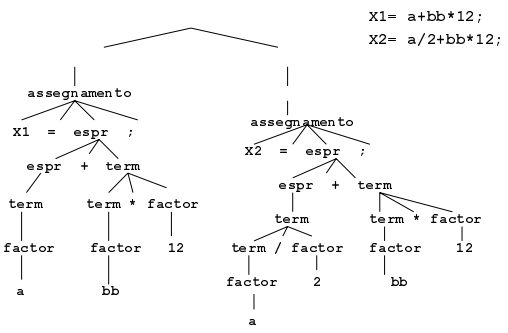
14

ALBERI SINTATTICI



15

ALBERI SINTATTICI



16

ANALIZZATORE SEMANTICO

- Analizzatore semantico: riceve come ingresso l'albero sintattico generato dal parser.
- Fasi principali
  - CONTROLLO STATICO: vengono svolti i controlli sui tipi, dichiarazioni ecc…
  - AZIONI DA COMPIERE: associazione di routine semantiche associate agli operatori che specificano quali azioni compiere (esempio: tipo operandi conforme all'operatore)
  - GENERAZIONE DI RAPPRESENTAZIONE INTERMEDIA

17

ANALIZZATORE SEMANTICO

- Considera gli aspetti dipendenti dal contesto
- Esempio
- Se `I` non e' dichiarato `J=I*K;` e' un'istruzione illegale, anche se sintatticamente corretta.
- Altri esempi:
- controllo di tipo, il controllo che una variabile sia stata dichiarata, il corretto utilizzo degli indici negli array ecc (dipendono dal particolare linguaggio di programmazione).

18

FORMATO INTERMEDIO

Il compilatore, durante la traslazione, può creare una forma sorgente intermedia.

- Nella forma intermedia si ignorano alcuni dettagli tipici della macchina target.
- Alcune forme intermedie
  - Notazione polacca;
  - Notazione a n-tuple;
  - Alberi sintattici astratti;
  - Codice di una macchina astratta.

19

ANALIZZATORE SEMANTICO

- Esempio **`X1= a+bb*12;`**
  - controlla che il tipo di X1, a e bb sia corretto rispetto agli operatori e al loro risultato. Inoltre, controlla le dichiarazioni delle variabili.
  - un esempio di rappresentazione intermedia può essere in forma a quadruple

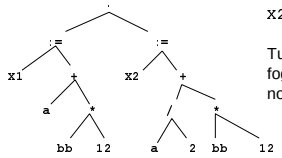
```
(*,B,12,R1)
(+,A,R1,X1)
```

20

ALBERO SINTATTICO ASTRATTO

- Un esempio di rappresentazione intermedia può essere quella che rimuove dall'albero sintattico alcune categorie intermedie e mantiene solo la struttura essenziale.

```
X1= a+bb*12;
X2= a/2+bb*12;
```



Tutti i nodi sono tokens. Le foglie sono operandi, mentre i nodi intermedi sono operatori

21

OTTIMIZZAZIONI

- Spesso a valle dell'analizzatorre semantico ci può essere un ottimizzatore del codice intermedio.
  - Propagazione di costanti

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

  evitando un accesso alla memoria

  - Eliminazione di sottoespressioni comuni

```
A=B*C;        T=B*C;
D=B*C;        A=T;
              B=T;
```

22

ANALISI: Riassumendo….

- Il compilatore nel corso dell'analisi del programma sorgente verifica la correttezza sintattica e semantica del programma:
  - ANALISI LESSICALE verifica che i simboli utilizzati siano legali cioè appartengano all'alfabeto
  - ANALISI SINTATTICA verifica che le regole grammaticali siano rispettate
  - ANALISI SEMANTICA verifica i vincoli imposti dal contesto

23

GENERATORE DI CODICE

- Generatore di codice: trasla la forma intermedia in linguaggio assembler o macchina
- Prima della generazione di codice:
  - ALLOCAZIONE DELLA MEMORIA
  - ALLOCAZIONE DEI REGISTRI

24

GENERATORE DI CODICE

- Istruzioni pseudo-assembler per una macchina di nostra invenzione

```
                                    X1:= a+bb*12;
PUSHADDR X2                         X2:= a/2+bb*12;
PUSHADDR X1
PUSH bb
PUSH a
LOAD R1 1(S)       mette bb in R1
MPY  12 R1         mette bb*12 in R1
LOAD R2 S          mette a in R2
STORE R2 R3        copia R2 in R3
ADD R1 R3          mette a+b*12 in R3
STORE @2(S)        mette a+b*12 in X1
DIV  2 R2          mette a/2 in R2
ADD R1 R2          mette a/2+b*12 in R2
STORE @3(S)        mette a/2+b*12 in X2
```

25

OTTIMIZZATORE DI CODICE

- Il codice generato può essere ottimizzato:
  - ottimizzazioni indipendenti dalla macchina
    - ad esempio rimozione di invarianti di ciclo, rimozione di espressioni duplicate
  - ottimizzazioni dipendenti dalla macchina
    - che riguardano ad esempio l'ottimizzazione sull'uso dei registri

26

COMPILATORI

- Abbiamo visto tutti i passi effettuati da un compilatore separatamente. In realtà questi passi possono essere uniti.
  - Scanner e parser possono essere eseguiti in sequenza, oppure lo scanner può essere chiamato dal parser ogni volta che necessita di un nuovo token.
  - Parser e analizzatore semantico possono essere uniti.
- Altri aspetti:
  - error detection e recovery
  - tabelle dei simboli generate dai vari moduli
  - gestione della memoria

27