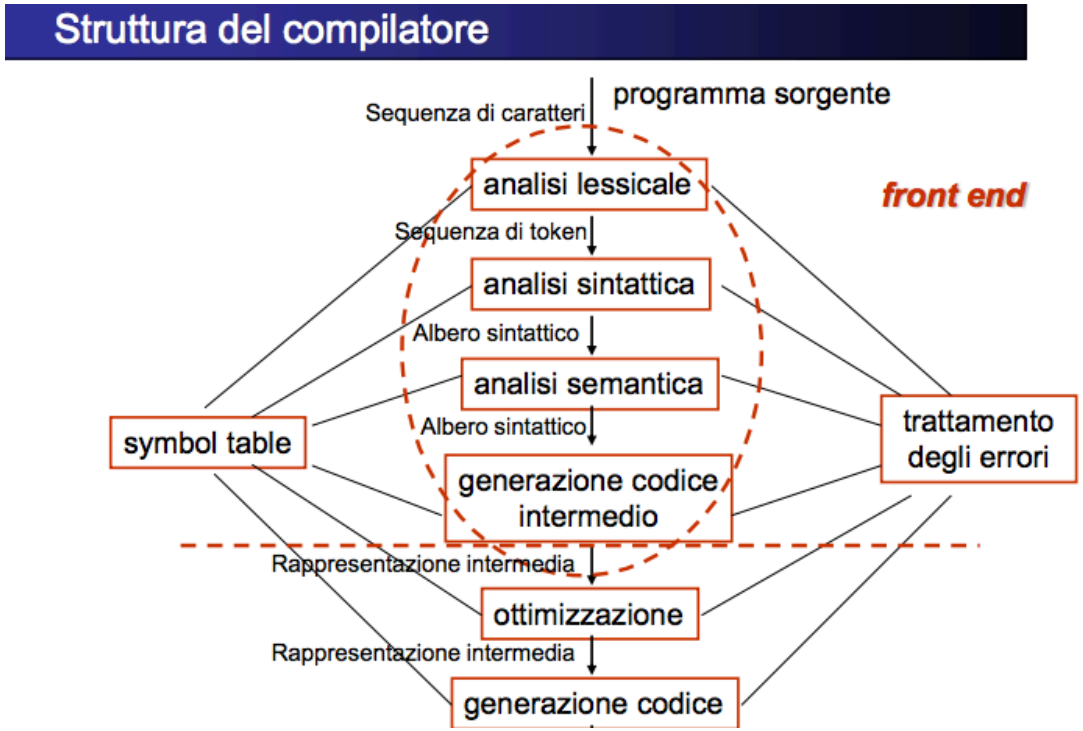
5. Generazione del bytecode

Su espressioni di un linguaggio di programmazione semplice (grammatica in 3.2)

Generazione di codice intermedio

- Dove si posiziona in una pipeline di traduzione?



Generazione del bytecode

- Obiettivo: realizzare un traduttore per i programmi scritti nel linguaggio di programmazione semplice dell'esercizio 3.2 e nella parte teorica del corso
  - che chiameremo di qui in avanti P.
- I file di programmi del linguaggio P hanno estensione .pas.
- Il traduttore deve generare bytecode [4] eseguibile dalla Java Virtual Machine (JVM).

Generazione di codice intermedio

- Generare bytecode eseguibile direttamente dalla JVM non è una operazione semplice
  - complessità del formato dei file .class (che tra l'altro e' un formato binario)
- Il bytecode verrà quindi generato avvalendoci di un linguaggio mnemonico (linguaggio assembler) che fa riferimento alle istruzioni della JVM e useremo come linguaggio target
- Il codice tradotto nel linguaggio target verrà tradotto successivamente nel formato .class dal programma assembler Jasmine, che effettua una traduzione 1-1 delle istruzioni mnemoniche nella corrispondente istruzione binaria della JVM.

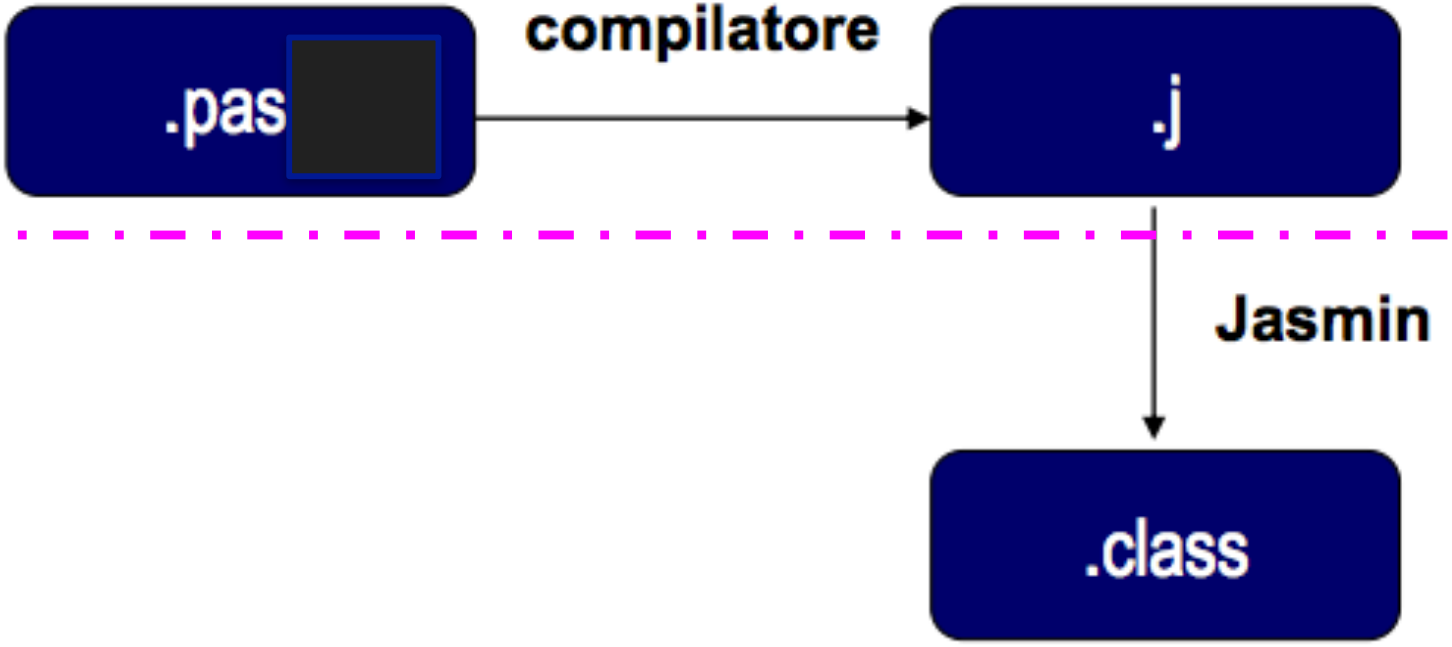
Generazione di codice intermedio

Codice in P:
linguaggio sorgente

```
print (10 + 20 * 30)
```

Codice in linguaggio assembler:
linguaggio target

```
ldc 10
ldc 20
ldc 30
imul
iadd
invokestatic Output/print(I)V
```



Generazione di bytecode

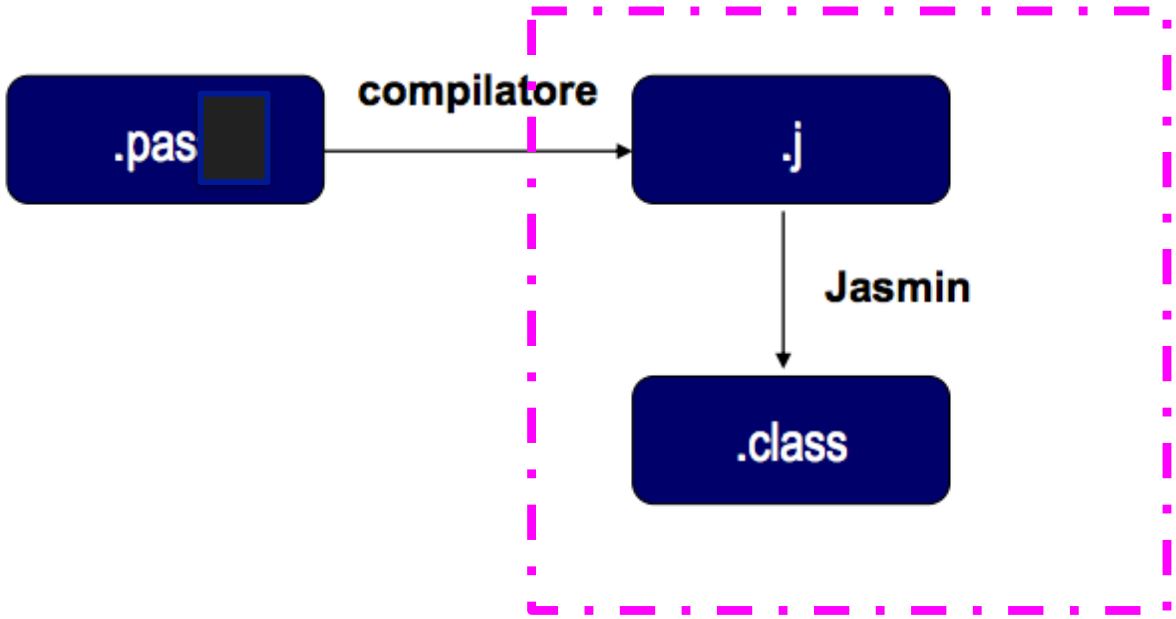
- Il linguaggio mnemonico utilizzato fa riferimento all'insieme delle istruzioni della JVM [5] e l'assembler effettua una traduzione 1-1 delle istruzioni mnemoniche nella corrispondente istruzione (opcode) della JVM.
- Il programma assembler che utilizzeremo si chiama Jasmin.
  - Distribuzione e manuale sono disponibili all'indirizzo http: // jasmin.sourceforge.net/
- La costruzione del file .class a partire dal sorgente scritto nel linguaggio P avviene secondo lo schema seguente

Generazione di bytecode

- Nel codice presentato in questa sezione, il file generato dal compilatore si chiama Output.j , e il comando

  java -jar jasmin.jar Output.j

- è utilizzato per trasformarlo nel file Output.class, che può essere eseguito con il comando
  - java Output
- Moodle: trovate jasmin.jar + URL Jasmin

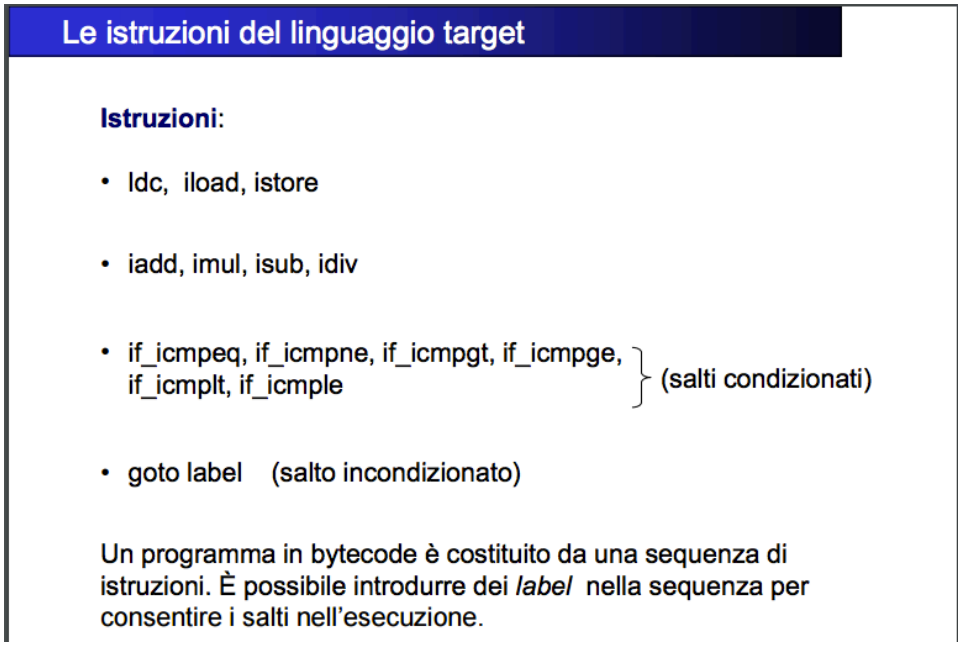


Classi di supporto

- Su Moodle trovate alcune classi di supporto già implementate:
- OpCode: semplice enumerazione dei nomi mnemonici
- Instruction: verrà usata per rappresentare singole istruzioni del linguaggio mnemonico.
  - Il metodo toJasmin restituisce l'istruzione nel formato adeguato per l'assembler Jasmin
- CodeGenerator: ha lo scopo di memorizzare in una struttura apposita la lista delle istruzioni (come oggetti di tipo Instruction) generate durante la parsificazione.
  - I metodi emit sono usati per aggiungere istruzioni o etichette di salto nel codice.
  - Le costanti header e footer definiscono il preambolo e l'epilogo del codice generato dal traduttore per restituire, mediante il metodo toJasmin, un file la cui struttura risponde ai requisiti dell'assembler Jasmin.
- SymbolTable: per tenere traccia degli identificatori occorre predisporre una tabella dei simboli.

Classi di supporto

- Su Moodle trovate alcune classi di supporto
  - OpCode: semplice enumerazione dei nomi mnemonici



Esercizio 5.1

- Per tenere traccia degli identificatori occorre predisporre una tabella dei simboli.
- Classe di supporto supplementare:
  - possibile implementazione della classe SymbolTable

```
public class SymbolTable {

    Map <String, Integer> OffsetMap = new HashMap <String,Integer>();

    public void insert( String s, int address ) {
            if( !OffsetMap.containsValue(address) )
                OffsetMap.put(s,address);
            else
                throw new IllegalArgumentException("Riferimento ad una
                   locazione di memoria gia` occupata da un'altra variabile");
    }

    public int lookupAddress ( String s ) {
            if( OffsetMap.containsKey(s) )
                return OffsetMap.get(s);
            else
                return -1;
    }
}
```

Esercizio 5.1

- Si scriva un traduttore per programmi scritti nel linguaggio P
  - utilizzate uno dei lexer sviluppati per gli esercizi di Sezione 2.
  - Fate riferimento alla grammatica del linguaggio P nell'esercizio 3.2.
- Su Moodle trovate un frammento del codice da completare di una possibile implementazione (che riguarda la gestione di print , read ,
  == (in `b_expr(int ltrue, int lfalse)`) e
  +

  - Translator.java
  - si noti che code è un oggetto della classe CodeGenerator

Esercizio 5.1

- Main

```
public static void main(String[] args) {
    Lexer_id_commenti lex = new Lexer_id_commenti();

    String path = "..URL..";
    try {
        BufferedReader br = new BufferedReader(new FileReader(path));
        Translator translator = new Translator(lex, br);
        translator.prog();
        br.close();
    } catch (IOException e) {e.printStackTrace();}
}
```

Esempi

- Semplici programmi P affiancati dal bytecode JVM corrispondente
- Precisazione su etichette

Listing 14: `B.pas`

```
read(a);
print(a+1)
```

Listing 15: Bytecode di `B.pas`

```
 invokestatic Output/read()I
 istore 0
L1:
 iload 0
 ldc 1
 iadd
 invokestatic Output/print(I)V
L2:
L0:
```

Listing 16: `C.pas`

```
x = 10;
y = 20;
z = 30;
print (x + y * z)
```

Listing 17: Bytecode di `C.pas`

```
 ldc 10
 istore 0
L1:
 ldc 20
 istore 1
L2:
 ldc 30
 istore 2
L3:
 iload 0
 iload 1
 iload 2
 imul
 iadd
 invokestatic Output/print(I)V
L4:
L0:
```

Precisazioni (1)

- Semantica dell'istruzione read(ID)
- il comando indicato nella grammatica con read(ID) permette l'inserimento di un numero intero dalla tastiera e l'assegnamento del valore del numero all'identificatore scritto tra parentesi.
  - Esempio

    ```
    read(a)
    ```

    specifica che l'utente del programma scritto in linguaggio P deve inserire un numero intero con la tastiera, che poi e' assegnato all'identificatore a (symbol table).

Precisazioni (2)

Equivale a una cascata di if else
Simile al costrutto switch case in Java)

- case when(⟨bexpr_1 ⟩)⟨stat_1 ⟩ · · ·
  when(⟨bexpr_n ⟩)⟨stat_n ⟩
  else ⟨stat ⟩
- Istruzione condizionale: dall'elenco di espressioni booleane ⟨bexpr_1 ⟩ · · · ⟨bexpr_n ⟩, se la prima espressione che risulta valutata come vero e` ⟨bexpr_k⟩ allora ⟨stat_k⟩ e' eseguito, si esce dall'istruzione case e si procede alla prossima istruzione. Se nessuna espressione nell'elenco ⟨bexpr_1⟩···⟨bexpr_n⟩ e' verificata, viene eseguito lo ⟨stat⟩ scritto dopo la parola chiave else.
- Ogni ⟨stat_k⟩ può essere una singola istruzione oppure una sequenza di istruzioni racchiusa tra parentesi graffe; la stessa cosa vale per ⟨stat⟩.
- Esempio:

```
case when(x>0) {print(x);print(y)} when(y>0) print(y) else print(z)
```

Richiamo teoria

Traduzione ‘on-the-fly’

x = 49 ; y = 21 ; `while` x ! = y `do` `if` x < y `then` y = y – x `else` x = x – y EOF

```
    ldc 49
    istore(addr(ID.x)
L3  ldc 21
    istore(addr(ID.y)
L2
L4  iload(addr(ID.x)
    iload(addr(ID.y)
    if_cmpne L5
    goto L1
L5  iload(addr(ID.x)
    iload(addr(ID.y)
    if_cmplt L6
    goto L7
L6  iload(addr(ID.y)
    iload(addr(ID.x)
    isub
    istore(addr(ID.y)
    goto L4
L7  iload(addr(ID.x)
    iload(addr(ID.y)
    isub
    istore(addr(ID.x)
    goto L4
L1  stop
```

Schemi di traduzione: on-the-fly