

5. Generazione del bytecode

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

Orario: update

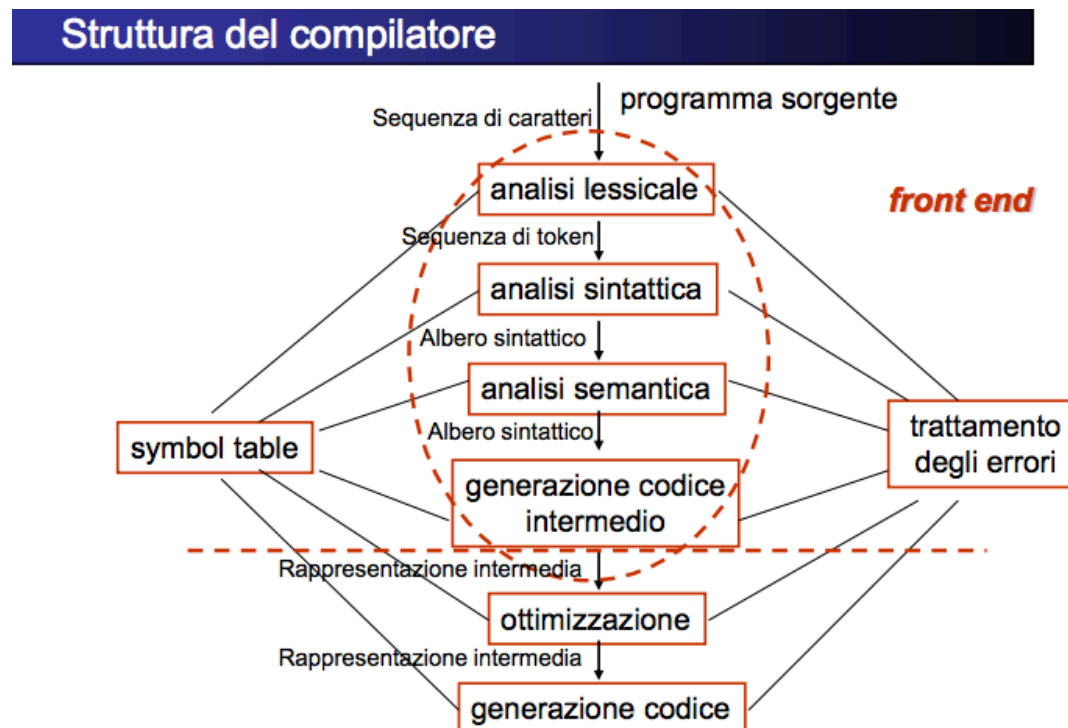
- Ultima lezione L10 a gennaio
- Mar 10 gennaio 2023:
13-16
- Aggiornate i calendari

January 2023

	Mon 9	Tue 10	Wed 11
all-day			
08:00			
09:00			
10:00			
11:00	11:00 LFT lab T3 - LINGUAG GI FORMAL...		
12:00			
13:00		13:00 LFT lab T1 - LINGUAG GI FORMAL...	13:00 LFT lab T3 - LINGUAG GI FORMAL...
14:00			
15:00			
16:00			
17:00			
18:00			

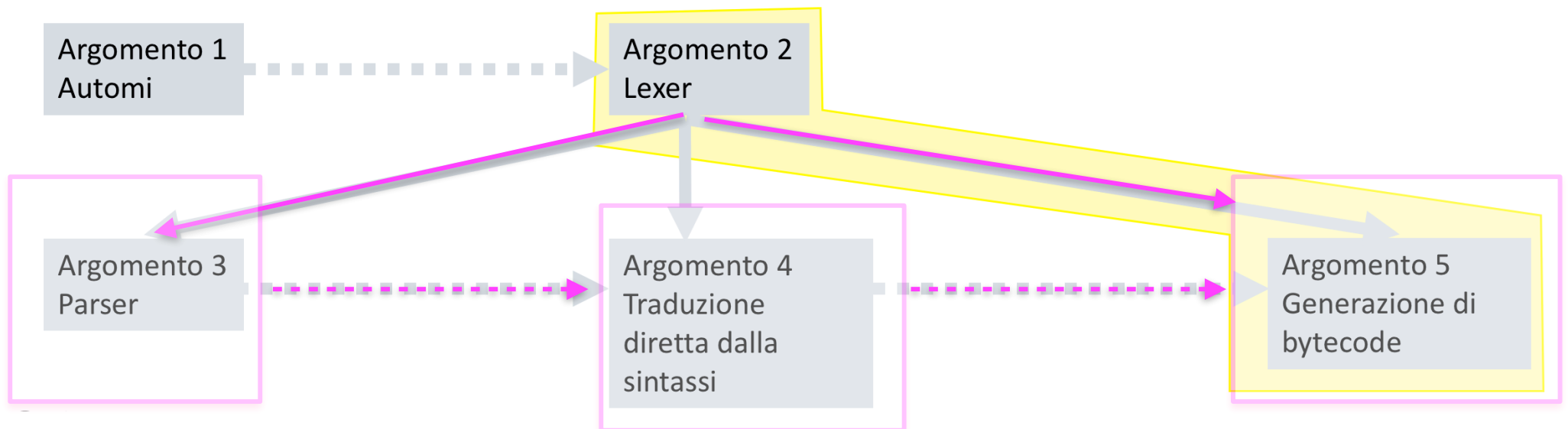
Generazione di codice intermedio

- Dove si posiziona in una pipeline di traduzione?



Progetto di laboratorio LFT LAB

- Il progetto di laboratorio consiste in una serie di **esercitazioni** assistite mirate allo sviluppo di un semplice **traduttore**.
- Dove siamo:
 - Quarto step: **Generazione del Bytecode**



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 **.lft**
- 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 è 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 (opcode) della JVM.

Generazione di codice intermedio

Codice in P:
linguaggio sorgente

Listing 13: Programma A.1ft

```
read[a];  
print[+(a,1)]
```

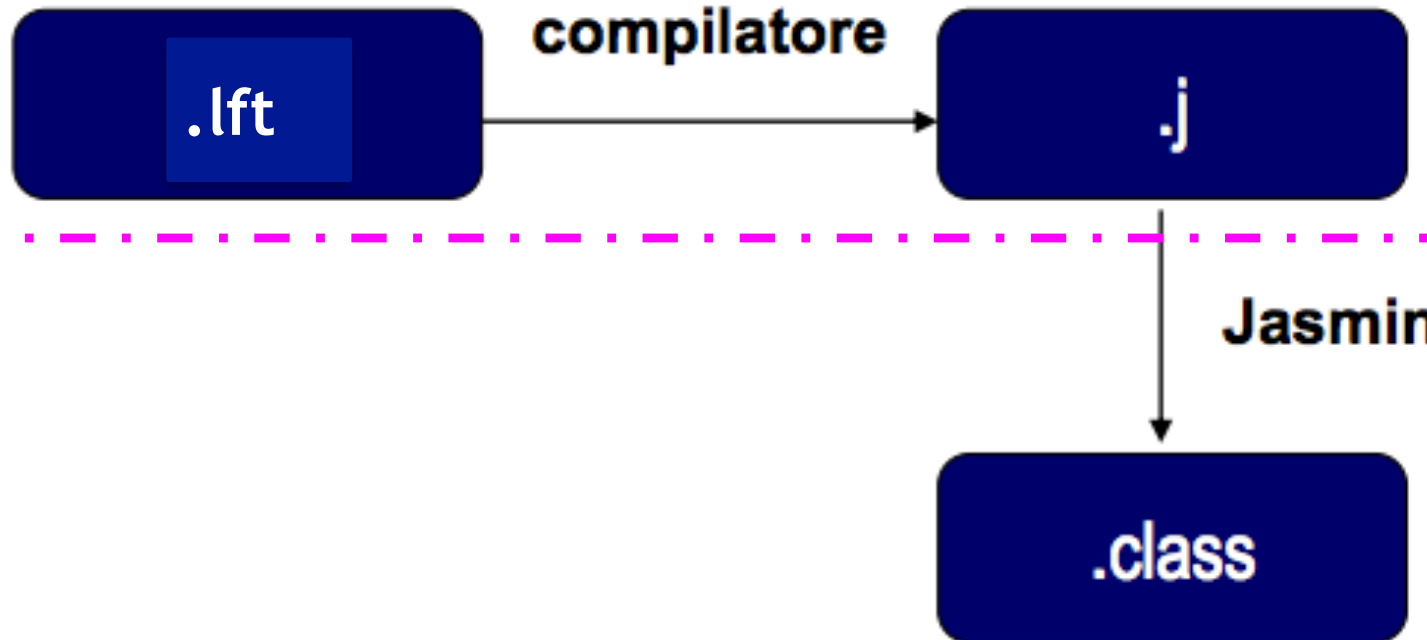
A.1ft

Codice in linguaggio assembler:
linguaggio target

Listing 14: Esempio di bytecode di A.1ft

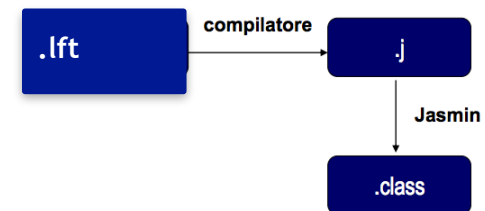
```
invokestatic Output/read() I  
istore 0  
goto L1  
L1:  
iload 0  
ldc 1  
iadd  
invokestatic Output/print(I)V  
goto L2  
L2:  
goto L0  
L0:
```

Bytecode di A.1ft



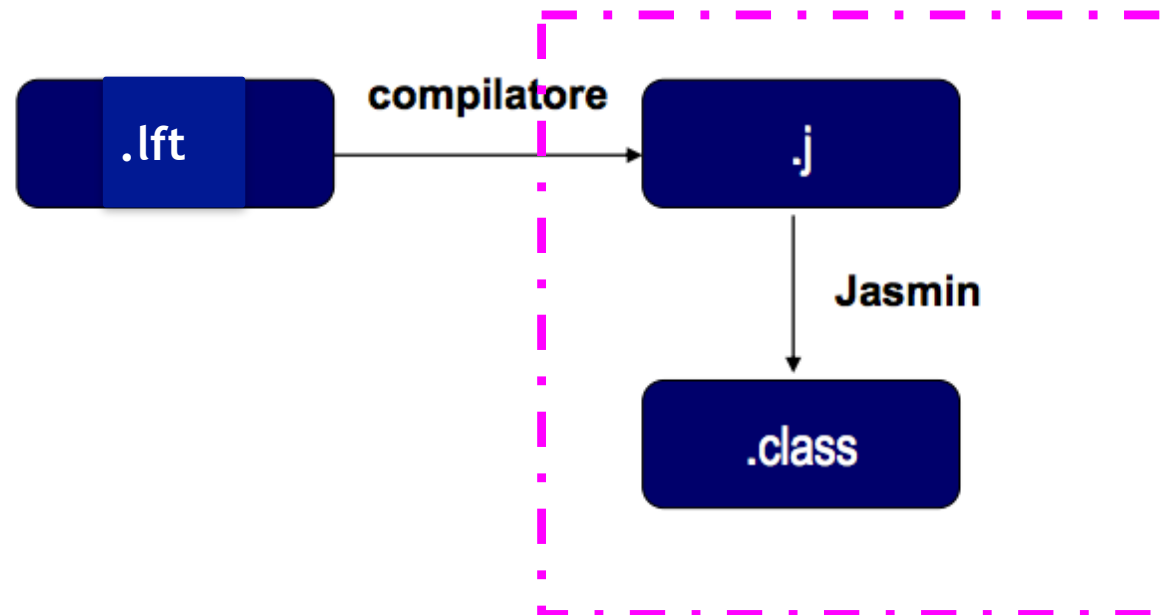
Generazione di bytecode

- 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/>
- Ricordando che la costruzione del file **.class** a partire dal sorgente scritto nel linguaggio P avviene secondo lo schema:



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**



Precisazioni - comandi del linguaggio (1)

- Semantica dell'istruzione `read [<idlist >]`
- Il comando indicato nella grammatica con `read [<idlist >]` permette l'inserimento di n numeri interi non-negativi dalla tastiera; l'*i*-esimo numero inserito dalla tastiera è assegnato all'identificatore ID_*i*. Esempio il comando:

`read[a,b]`

specifica che l'utente del programma scritto in linguaggio P deve **inserire due numero interi non negativi da tastiera**, il primo dei quale è assegnato all'identificatore **a** e il secondo dei quali è assegnato all'identificatore **b** (symbol table)

Precisazioni - comandi del linguaggio (2)

conditional [option($\langle bexpr_1 \rangle$) do $\langle stat_1 \rangle \dots$ option($\langle bexpr_n \rangle$) do $\langle stat_n \rangle$] end

Un comando condizionale: dall'elenco di espressioni booleane $\langle bexpr_1 \rangle \dots \langle bexpr_n \rangle$, se la prima espressione che risulta valutata come vera è $\langle bexpr_k \rangle$ allora $\langle stat_k \rangle$ è eseguito, si esce dall'istruzione conditional e si procede alla prossima istruzione. Si noti che ogni $\langle stat_k \rangle$ può essere una singola istruzione oppure una sequenza di istruzioni racchiusa tra parentesi graffe. Se nessuna espressione nell'elenco $\langle bexpr_1 \rangle \dots \langle bexpr_n \rangle$ è verificata, si procede alla prossima istruzione dopo l'istruzione conditional. Ad esempio, l'istruzione

```
conditional [ option(> x 0) do print[x,y]
              option(> y 0) do print[y]
            ] end
```

stampa sul terminale i valori sia di x e y se il valore di x è maggiore di 0, e solo il valore di y se il valore di x è uguale o inferiore a 0 e quello di y è maggiore di 0.

conditional [option($\langle bexpr_1 \rangle$) do $\langle stat_1 \rangle \dots$ option($\langle bexpr_n \rangle$) do $\langle stat_n \rangle$] else $\langle stat \rangle$ end

Aggiunge al comando condizionale precedente un caso simile a *else* o *default* di Java: lo $\langle stat \rangle$ scritto dopo la parola chiave *else* viene eseguito nel caso in cui nessuna espressione booleane nell'elenco $\langle bexpr_1 \rangle \dots \langle bexpr_n \rangle$ risulta valutata come vera. Più precisamente, se la prima espressione booleana che risulta valutata come vera è $\langle bexpr_k \rangle$ allora $\langle stat_k \rangle$ è eseguito, si esce dall'istruzione conditional e si procede alla prossima istruzione; invece se nessuna espressione nell'elenco $\langle bexpr_1 \rangle \dots \langle bexpr_n \rangle$ è verificata, viene eseguito lo $\langle stat \rangle$ scritto dopo la parola chiave *else*, dopo di che si procede alla prossima istruzione. Ad esempio, l'istruzione

```
conditional [ option(> x 0) do print[x,y]
              option(> y 0) do print[y]
            ] else print[z] end
```

stampa sul terminale i valori sia di x e y se il valore di x è maggiore di 0, solo il valore di y se il valore di x è uguale o inferiore a 0 e quello di y è maggiore di 0, e solo il valore di z se i valori sia di x che di y sono uguali o inferiori a 0.

Precisazioni - comandi del linguaggio (2)

while (*<bexpr>*) *<stat>*

Permette l'esecuzione ciclica di *<stat>*. La condizione per l'esecuzione del ciclo è *<bexpr>*. Si noti che *<stat>* può essere una singola istruzione oppure una sequenza di istruzioni racchiusa tra parentesi graffe. Ad esempio, l'istruzione

```
while (> x 0) { assign - x 1 to x; print[x] }
```

decrementa e stampa sul terminale il valore dell'identificatore x, fino a quando x raggiunge il valore 0.

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

Le istruzioni del linguaggio target

Istruzioni:

- ldc, iload, istore
- iadd, imul, isub, idiv
- if_icmpeq, if_icmpne, if_icmpgt, if_icmpge, if_icmplt, if_icmple } (salti condizionati)
- goto label (salto incondizionato)

Un programma in bytecode è costituito da una sequenza di istruzioni. È possibile introdurre dei *label* nella sequenza per consentire i salti nell'esecuzione.

Esercizio 5.1

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

L'indirizzo associato con un identificatore può essere utilizzato come argomento dei comandi **iload** oppure **istore**.

```
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

- Richiamo di teoria

L'indirizzo associato con un identificatore può essere utilizzato come argomento dei comandi **iload** oppure **istore**.

Struttura di un frame della JVM

Ogni frame corrisponde a un metodo in esecuzione e contiene:

- argomenti e variabili locali (indirizzati a partire da 0);
- pila degli operandi (cresce/cala durante l'esecuzione del metodo).

Nome	Slot n.	Valore
<i>a</i>	0	42
<i>b</i>	1	true
<i>x</i>	2	7
<i>y</i>	3	23
<i>z</i>	4	'c'
—	—	5
—	—	7
		⋮

```
static void m(int a, boolean b) {  
    int x, y;  
    char z;  
  
    ... 5 * x ...  
}
```

Nota

- Nei metodi non statici il primo argomento è il riferimento all'oggetto ricevente (*this*).

Gestione della pila degli operandi

Istruzione	Prima	Dopo	Descrizione
ldc <i>v</i>		<i>v</i>	carica <i>v</i> sulla pila
iload <i>&x</i>		<i>v</i>	carica il valore di <i>x</i> sulla pila
istore <i>&x</i>		<i>v</i>	assegna <i>v</i> a <i>x</i>
pop		<i>v</i>	rimuove il valore in cima alla pila
dup		<i>v v</i>	duplica il valore in cima alla pila
swap	<i>v₁ v₂</i>	<i>v₂ v₁</i>	scambia i due valori in cima alla pila

Note

- Il valore in cima alla pila è quello più a destra.
- Le istruzioni **iload** e **istore** hanno come argomento l'indirizzo – e non il nome – della variabile *x* nel frame del metodo corrente.

Riflessioni

- Inizializzazione di valori di variabili e eventuali errori da segnalare a livello di traduzione del bytecode

Esercizio 5.1

- Si scriva un traduttore per programmi scritti nel linguaggio P
 - utilizzate uno dei lexer sviluppati per gli esercizi di Sezione 2 (possibilmente l'ultimo che include la possibilità di riconoscere i commenti e di specificare identificatori che includano underscore).
 - 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 read (in <stat> e in <idlist>) e - (in expr ())
 - 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 13: Programma A. lft

```
read[a];  
print[+(a, 1)]
```

Listing 14: Esempio di bytecode di A. lft

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

Listing 15: Programma B. lft

```
assign 10 to x;  
assign 20 to y;  
assign 30 to z;  
print[+(x, *(y, z))]
```

Listing 16: Esempio di bytecode di B. lft

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

Testing

- Potete trovare alcuni esempi su cui testare il funzionamento del traduttore nella sezione della settimana del 15-21 novembre.
- Siete incoraggiati a scrivere anche altri esempi e utilizzarli per testare il traduttore.
- Più esempi testate con diverse caratteristiche e costrutti, meglio è!



Precisazioni (3)

- Ricordate che le operazioni di somma e moltiplicazione possono avere un numero n di argomenti con $n \geq 1$.
- Invece, le operazioni di sottrazione e divisione hanno esattamente due argomenti.
- Ad esempio:
 - l'espressione: $*(2\ 3\ 4)$ ha valore 24
 - l'espressione $-(2\ 4)\ 3$ ha valore 5
 - l'espressione $+(2 - 7\ 3)$ ha valore 6
 - l'espressione $+(/\ 10\ 2\ 3)$ ha valore 8
 - l'espressione $+(5 - 7\ 3\ 10)$ ha valore 19

FAQ

- Domanda
- Dato che $\langle \text{exprlist} \rangle$ può corrispondere a una lista con un singolo elemento, qual è il significato di un'espressione con $+$ e $*$ e con un singolo operando, ad esempio $+(3)$ oppure $*(5)$?
- Risposta
- Possiamo adottare la convenzione seguente: il valore di un'espressione con $+$ oppure $*$ e con un singolo operando è uguale al valore dell'operando. Ad esempio, il valore di $+(3)$ è uguale a 3, e il valore di $*(5)$ è uguale a 5.