Contents

1 Idea generale **1**

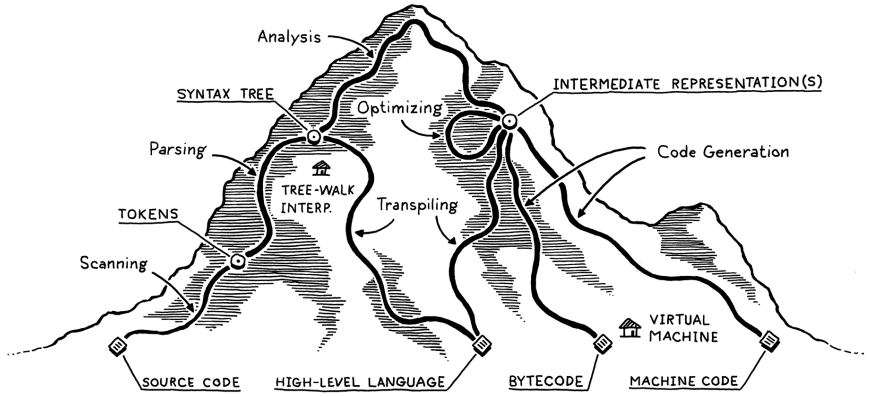
2 Prendi il codice **2**

3 Struttura del codice **3**
3.1 Tokenizing 3
3.2 Il resto 4
3.2.1 Perché è una cazzata farlo in lisp 4
3.2.2 Perchè non è una cazzata farlo in java 4

4 Esecuzione della struttura **5**

1 Idea generale

ci sono molti modi di interpretare/compilare/eseguire un linguaggio per fottere un attimo i disegni a questo libro[1]



quello che intendo fare, per mera facilità di interpretazione, è un `Tree-Walk Interpreter` (quella casetta messa a cazzo in mezzo alla montagna)
sarebbe a dire

- prendi il codice
- capisci che cazzo di struttura ha

[1] di cui ho letto mezzo capitolo prima di rompermi il cazzo che non potevo scaricare il pdf, perchè sono un bambino viziato

- runna brutalmente dalla struttura senza neanche pensarci di usarla per transpiling &Co.

ognino di questi passi ha come input l'output dello step precedente

- la prima parte ha come output il codice
- capire la struttura ha come input il codice e come output la struttura
- runnare la struttura ha come input la struttura

quindi rappresentiamo tutto malissimo con un

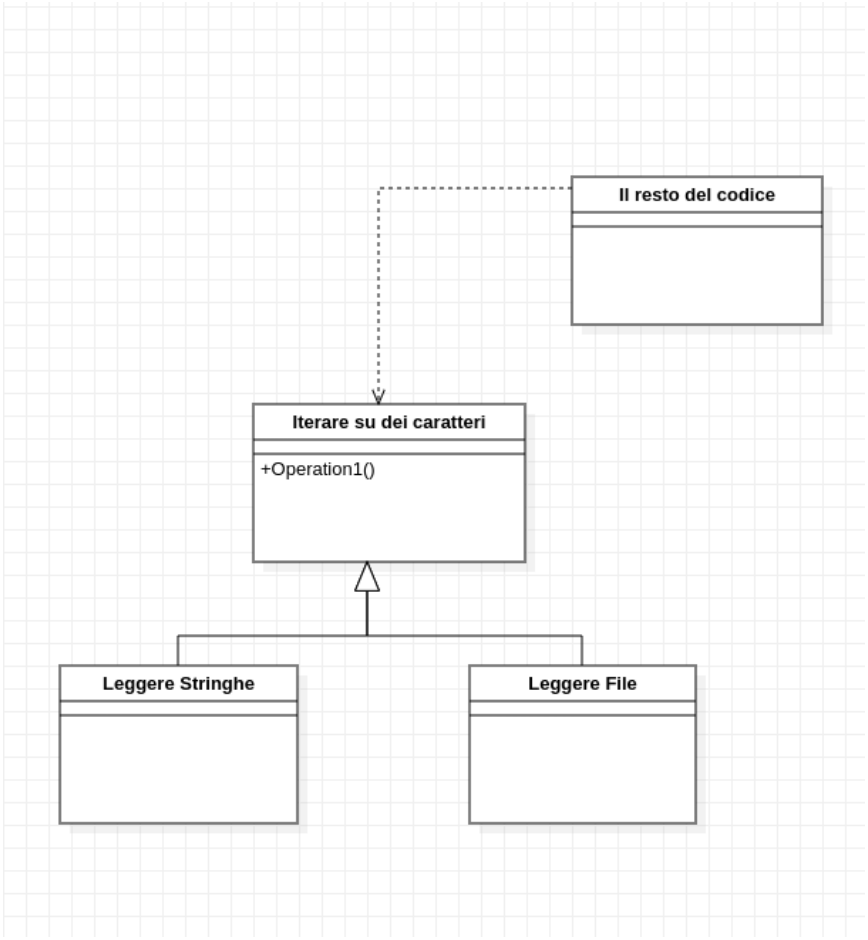
```
codice -> struttura -> esecuzione struttura
```

si illustrano mo' queste fasi del programma

2 Prendi il codice

abbiamo due modi per prendere il codice, il primo è il classico "leggi un file" a fini di testing e per certi use case vorrei però che fosse anche possibile leggere codice lisp da una stringa messa nel codice java

per abusare un po' l'uml possiamo metterla in questo modo



3 Struttura del codice

per semplificarci la vita, capire la struttura del codice qui si divide in 2 passi

- tokenizing
- il resto

3.1 Tokenizing

con tokenizing si intende prendere la stringa di codice e dividerla in quei pezzi che hanno effettivamente un significato, quindi un

```
"for(int i=1; i<max; ++i) {res = res*i;}"
```

viene diviso nella sequenza

```
{"for", "(", "int", "i", "=", "1", ";", "i", "<", "max", ";", "++", "i",
↪  ")",
        "{", "res", "=", "res", "*", "i", ";", "}"}
```

3.2 Il resto

il resto consiste nel prendere la sequenza di sottostringhe data dal tokenizer e usarla per costruire il dannatissimo *Albero del Codice*.

3.2.1 Perché è una cazzata farlo in lisp

qui è dove lisp inizia a tirare di brutto, perchè l'albero del codice è definito solo ed interamete dalle parentesi, per quanto "prendi stringa di parentesi e costruisci l'albero che definisce" sia un po' stronzo, non è niente rispetto a quanto ci vuole a fare parsing di linguaggio più comuni.

3.2.2 Perchè non è una cazzata farlo in java

se pensi in java potresi avere anche solo un

```
x + 2;
```

che ti obbligherebbe a dover

- capire che `x + 2` è un'espressione
- capire che è l'applicazione dell'operatore `+` a `x` e `2`

non troppo bastardo, ma metti un

```
x + 2 * y;
```

qui dobbiamo

- capire che è un'espressoine
- capire che `+` e `*` sono gli operatori
- andare di precedenza di operatori per fare `x + (2 * y)`

visto che ho già abbastanza traumi di mio con la precedenza di operatori, e che non voglio capire come implementarla, direi di evitare

poi andiamo a casi come

```
if(x>10)
    return x;
else
    return x+2;
```

e onestamente, direi di evitare

4 Esecuzione della struttura

> **NOTA**: di solito per capire un minimo cosa cazzo sto facendo devo iniziare a implementarlo. Questa parte non l'ho ancora fatta, quindi la discussione potrebbero essere un po' confusa. C'è evaluator.org nella cartella docs/ (quella sopra questa) con un po' di dettagli extra, ma sono più note che mi sono fatto per me, quindi potrebbe far cagare altrettanto il cazzo, scusate

ora che abbiamo la struttura (di lista) del codice possiamo pensare a come eseguirla, prendiamo, come esempio di questa sezione, l'espressione

```
(if (= x 10)
    (print "fa 10")
  (print "non fa 10"))
```

per eseguire questa espressione dobbiamo

- capire che è un `if`
- ora che sappiamo che è un `if`
  - capire quale sia la condizione da controllare
  - capire quale parte del `then`
  - capire quale quella dell'`else`
- determinare se la condizione da controllare è vera
  - se lo è esegui la parte del `then`
  - altrimenti quella dell'`else`

detta in modo un po' meno stronzo

- capire che è un `if`
- capito che è un `if`, determinare le parti dell'`if`, "costruire" l'`if`, per così dire

- costruito l'if, eseguirlo

sono sotto illustrate le tre parti in questione

> sotto non gli ho ancora illustarati perchè devo capire meglio come farli e come documentarli