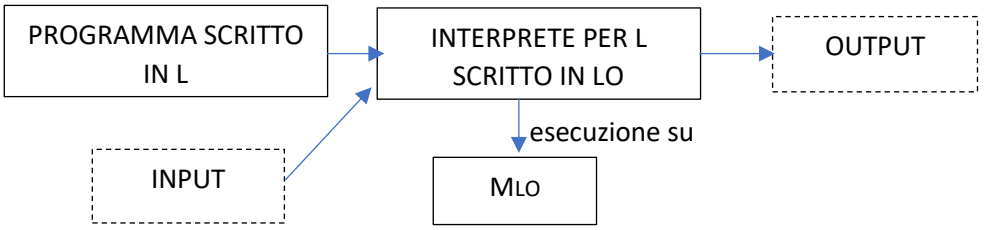
DOMANDE DI TEORIA

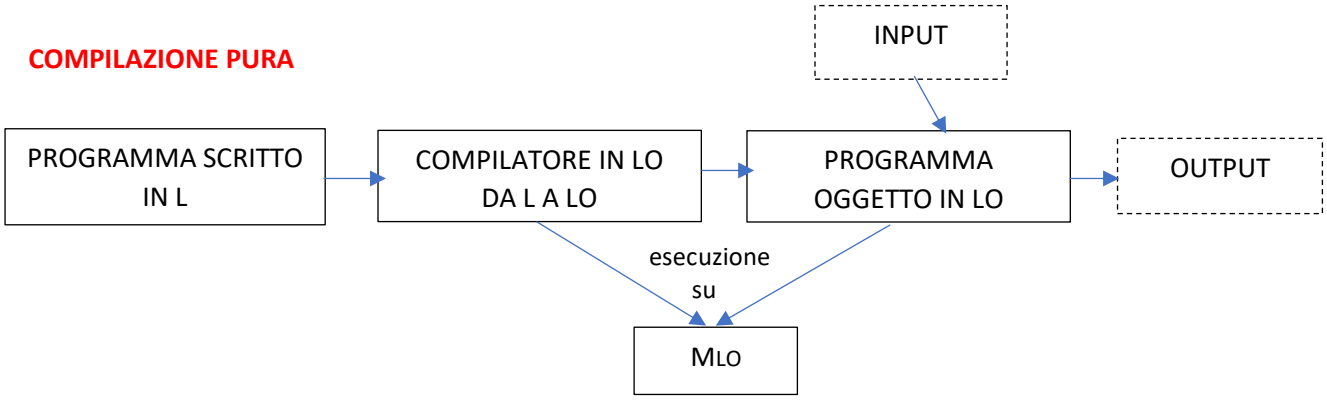
1. Il primo Fortran supportava la ricorsione. **FALSO**
2. Il primo Fortran supportava un'operazione analoga a una malloc. **FALSO**
3. Nel primo Fortran l'occupazione di memoria di un programma era nota a tempo di compilazione. **VERO**
4. Il primo Fortran aveva uno heap e uno stack di attivazione. **FALSO**
5. Il LISP è un linguaggio imperativo. **FALSO**
6. Il Prolog puro supporta i cicli for/next. **FALSO**
7. In Prolog la ricerca del massimo in una lista può essere fatta in forma iterativa (non ricorsiva). **FALSO**
8. Il predicato Prolog member può enumerare i membri di una lista e generare liste. **VERO**
9. SQL è un linguaggio general purpose. **FALSO**
10. HTTP (senza scripts) è un linguaggio general purpose. **FALSO**
11. Un linguaggio puramente funzionale ha i cicli while. **FALSO**
12. Nel paradigma funzionale si possono usare cicli for e while. **FALSO**
13. Nel paradigma logico non si distinguono parametri di input e output. **VERO**
14. Nel paradigma funzionale puro non ci sono gli assegnamenti. **VERO**
15. La funzione membro (elemento, lista) nel paradigma funzionale può essere usata anche per generare tutti gli elementi della lista. **FALSO**
16. Il C++ ha un garbage collector. **FALSO**
17. Il primo garbage collector è stato promosso nel LISP. **VERO**
18. Nei linguaggi funzionali env(x) restituisce il valore di x. **VERO**
19. Nei linguaggi funzionali env(x) restituisce una locazione. **FALSO**
20. In C, l'identificatore x in y=x rappresenta env(x). **FALSO**
21. In C, l'identificatore &x in y=&x rappresenta env(x). **VERO**
22. In un linguaggio funzionale puro, un identificatore x rappresenta env(x). **VERO**
23. Nei linguaggi funzionali l'ambiente associa direttamente gli identificatori al loro valore. **VERO**
24. Nei linguaggi imperativi l'ambiente associa direttamente gli identificatori al loro valore. **FALSO**
25. I linguaggi funzionali non sono computazionalmente completi. **FALSO**
26. In un linguaggio fortemente tipato il controllo dei tipi avviene durante la compilazione e l'esecuzione. **VERO**
27. In un linguaggio staticamente tipato il controllo dei tipi avviene sempre interamente a tempo di compilazione ed esecuzione. **VERO**
28. In un linguaggio dinamicamente tipato il controllo dei tipi avviene interamente a tempo di esecuzione. **VERO**
29. C è debolmente tipato. **VERO**
30. Se il linguaggio è dinamicamente tipato, allora il tipo di una variabile può cambiare durante l'esecuzione del programma. **VERO**
31. Il controllo di correttezza dei downcast richiede controlli a runtime. **VERO**
32. Un linguaggio fortemente e staticamente tipato può avere le union del C. **FALSO**
33. Il polimorfismo per inclusione è il più indicato per la definizione di collezioni di oggetti omogenei. **FALSO**
34. Se in Java si usa unicamente il polimorfismo parametrico allora tutti i controlli di tipo avvengono a tempo di compilazione. **VERO**
35. Il polimorfismo che permette più controlli a tempo di compilazione è quello parametrico. VERO
36. In C, il sistema dei tipi adotta solo la name equivalence. **FALSO**
37. In C, il sistema dei tipi adotta solo la structural equivalence. **FALSO**
38. In C, il sistema dei tipi adotta la name equivalence e la structural equivalence. **VERO**
39. Il C adotta sempre la structural equivalence tra tipi. **FALSO**
40. Il C adotta la structural equivalence per le struct e la name equivalence per tutti gli altri tipi. **FALSO**
41. Si può accedere alle variabili non locali di una procedura in tempo costante, indipendentemente da quanti record di attivazione di devono attraversare. **VERO**
42. La JVM può caricare classi da host diversi. **VERO**
43. Il codice oggetto deve essere eseguito da un interprete diverso dalla macchina hardware. **FALSO**
44. Il polimorfismo parametrico puro può generare errori di tipo a runtime. **FALSO**

45. Le macro hanno un proprio ambiente locale implementato con un record di attivazione. **FALSO**
46. Nei linguaggi a oggetti l'ambiente non locale di un metodo si può trovare: **NELLO HEAP** e **NELLA ZONA STATICA**.
47. La dimensione degli oggetti nello heap può essere esponenziale nell'altezza della gerarchia delle classi (ovvero nel numero di superclassi di una data classe): **IN C++**.
48. Il comando new in Java alloca memoria: **NELLO HEAP**.
49. L'ambiente non locale di una classe interna ad un'altra classe si può trovare: **NELLO HEAP** e **NELLA ZONA STATICA**.
50. Un tipo di dato astratto è: **UN TIPO PERFETTAMENTE INCAPSULATO**.
51. Nei linguaggi funzionali env(x) restituisce: **IL VALORE DI X**.
52. In Java l'ambiente non locale dei metodi si può trovare: **NELLO HEAP** e **NELLA ZONA DOVE SONO MEMORIZZATE LE CLASSI**.
53. Quali forme di polimorfismo supporta Java: **AD HOC**, **PER INCLUSIONE** e **PARAMETRICO**.
54. In un programma che usa le union o altre forme di record varianti il controllo dei tipi può essere fatto interamente a tempo di compilazione: **FALSO**.

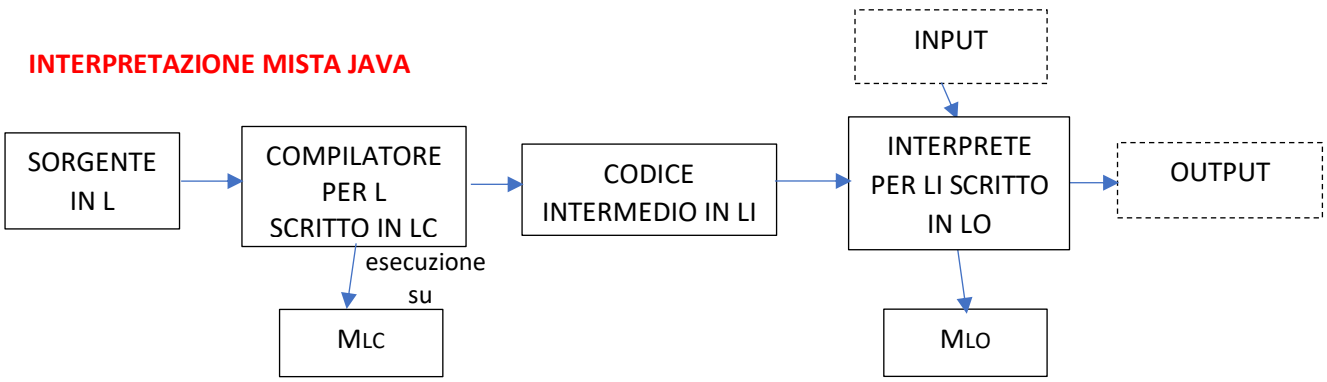
INTERPRETAZIONE PURA



COMPILAZIONE PURA



INTERPRETAZIONE MISTA JAVA



JAVA SU LINUX