Raccolta domande e risposte esami linguaggi di programmazione

<https://github.com/Pater999/UNITN-lingprog-simulatore-mod2>

2020

**NOTE**

In questo PDF sono contenute molte domande con risposta utili a preparare la parte teorica dell’esame del corso “Linguaggi di programmazione modulo 2” tenuto all’università di Trento dal professore Kuper. Le domande qui presenti sono state prese dagli esami passati e alcune di esse sono state inventate da me per coprire argomenti nuovi inseriti negli anni successivi.

Questa raccolta è aggiornata all’esame di **luglio 2022**.

Su queste domande è basato anche un **simulatore web** che permette di simulare il quiz d’esame. Trovate il sito web in html e Javascript che permette di simulare la parte teorica dell'esame a [QUESTO LINK](https://pater999.github.io/UNITN-lingprog-simulatore-mod2).

Il simulatore e questi appunti sono stati creati da [Pater999](https://t.me/Pater999).

La versione aggiornata di questi appunti è possibile trovarla su [github](https://github.com/Pater999/UNITN-lingprog-simulatore-mod2) dove è presente anche il codice sorgente del simulatore.

**Parte pratica (SML)**

Trovate del materiale utile a preparare la parte pratica dell’esame (programmino in SML) in questa [repository di github](https://github.com/Pater999/UNITN-linguaggi-programmazione-SML).

Sommario

[Domande generali teoria 0](#_Toc45230576)

[Domande lambda calcolo 9](#_Toc45230577)

[Domande calcolo indirizzo array 12](#_Toc45230578)

[Domande tipi funzioni ML 15](#_Toc45230579)

[Domande output codici 18](#_Toc45230580)

# **Domande generali teoria**

1. **La ricorsione in coda**
   * Non è implementabile nei linguaggi imperativi
   * **Nessuna delle altre risposte**
   * Richiede di non scrivere mai la chiamata ricorsiva come ultimo statement di una subroutine
   * Richiede di non ritornare mai direttamente il valore ritornato da una chiamata ricorsiva
   * Permette di risolvere il problema della ricorsione infinita
2. **Call by constant**
   * Fa una copia della variabile
   * **è utile per passare oggetti**
   * Ritorna il valore al chiamante
   * Permette la modifica della variabile
   * Nessuna delle altre risposte
3. **I record di attivazione**
   * Sono necessari solo in presenza di funzioni di ordine superiore
   * Sono allocati dinamicamente solo in caso di scope dinamico
   * **Nessuna delle altre risposte**
   * Devono essere esplicitamente allocati e deallocati dal codice del programma che li usa
   * Sono allocati solo nello heap
4. **Si può dire che una macchina astratta che capisce il linguaggio C non sia implementata in modo puramente compilativo perché**
   * Gli eseguibili generati da un compilatore C in genere non eseguono direttamente sulla macchina hardware, ma su una macchina astratta che include il Runtime del linguaggio e le funzionalità del Sistema Operativo
   * Gli eseguibili generati dal compilatore vengono comunque interpretati da una macchina virtuale
   * Una macchina astratta che capisca un linguaggio di alto livello come il C non è mai implementabile con un compilatore
   * Il Runtime del linguaggio C è comunque sempre interpretato
   * **Nessuna delle altre risposte**
5. **Il costrutto for dei linguaggi C, C++ e Java**
   * **Non è un costrutto di iterazione determinata**
   * È necessario a tali linguaggi per implementare qualsiasi tipo di algoritmo
   * Nessuna delle altre risposte
   * È un costrutto di iterazione determinata
   * Permette di sapere in anticipo quante volte il ciclo verrà ripetuto (indipendentemente dal corpo del ciclo)
6. **Un'entità esprimibile è**
   * Nessuna delle altre risposte
   * **Un'entità che può essere generata come risultato da un'espressione complessa o da una funzione**
   * Un'entità che ancora non compare nell'ambiente
   * Un'entità che può essere memorizzata
   * Una generica entità a cui può essere dato un nome
7. **In assenza di ambiente non locale**
   * Per implementare funzioni definite ricorsivamente è necessario utilizzare un fixed point combinator
   * Non si possono implementare algoritmi ricorsivi
   * **Nessuna delle altre risposte**
   * Non si possono implementare algoritmi iterativi o ricorsivi
   * Non si possono implementare algoritmi iterativi
8. **I puntatori di catena dinamica contenuti in un record di attivazione**
   * Collegano una lista di zone di memoria gestita dinamicamente
   * Servono per accedere alle variabili dinamiche
   * Non esistono “puntatori di catena dinamica” in un record di attivazione
   * Nessuna delle altre risposte
   * **Permettono, a partire da un RdA, di trovare il RdA precedente sullo stack**
9. **In presenza di variabili modificabili**
   * Nessuna delle altre risposte
   * **Esistono un Ambiente che associa valori denotabili (fra cui le locazioni di memoria) a nomi ed una Memoria che associa locazioni di memoria a valori memorizzabili**
   * Non esistono valori denotabili
   * La valutazione del comando di assegnamento restituisce sempre un valore
   * Il comando di assegnamento non ha effetti collaterali
10. **Il concetto di variabile modificabile**
    * È l'unico concetto utilizzabile quando si parla di variabili
    * È imposto dall'architettura di Von Neumann (variabili non modificabili richiederebbero macchine astratte caratterizzate da memoria a sola lettura)
    * Nessuna delle altre risposte
    * **È tipico del paradigma di programmazione imperativo**
    * Permette di evitare il fenomeno dell'aliasing
11. **Nella sostituzione (λa.abc)[arrg/c]**
    * **È necessario applicare una Alfa-equivalenza per evitare una cattura di variabile**
    * Viene catturata la variabile c
    * Si rischia di catturare la variabile “a” ed è necessario applicare Beta-equivalenza per risolvere in problema
    * Non c’è alcuna cattura di variabile
    * Nessuna delle altre risposte
12. **Il fenomeno della cattura di variabili**
    * Nessuna delle altre risposte
    * Non può essere evitato in alcun modo
    * **Fa sì che dopo una sostituzione una variabile libera diventi legata (per esempio da un’astrazione λx.)**
    * Comporta la “sparizione” di variabili libere durante un’astrazione funzione
    * È dovuto all’assenza di un ambiente non locale
13. **Una Macchina Astratta ML (LO) è**
    * È un modo per descrivere un interprete
    * Nessuna delle altre risposte
    * È implementabile solo basandosi sull’architettura di Von Neumann
    * È un modo per descrivere un compilatore
    * **È associata ad un proprio linguaggio macchina L, che è in grado di capire ed eseguire**
14. **Il passaggio di parametri per nome**
    * Permette di passare valori solo dal chiamante al chiamato (e non viceversa)
    * Nessuna delle altre risposte
    * **È implementabile passando una chiusura come parametro**
    * Ha un valore solo teorico e non è implementabile in pratica
    * Permette la cattura di variabili libere in modo da effetti non deterministici
15. **La tecnica del display**
    * Nessuna delle altre risposte
    * Permette di implementare facilmente lo scope dinamico
    * **Permette di ridurre il costo derivante dalla scansione della catena statica quando si implementa lo scope statico**
    * Permette di visualizzare le zone di memoria allocata dinamicamente
    * Permette di implementare le regole di scope statico senza generare frammentazione della memoria
16. **Un compilatore da un linguaggio L ad un linguaggio LO è**
    * **Un programma che trasforma un programma PL (espresso nel linguaggio L) in un programma PLO (espresso nel linguaggio LO) tale che per ogni input I si ha PL(I) = PLO(I)**
    * Nessuna delle altre risposte
    * Un programma scritto nel linguaggio LO che riceve come ingresso un programma PL (espresso nel linguaggio L) ed il suo input I generando lo stesso output che genera PL con input I
    * L’implementazione di una macchina astratta scritta nel linguaggio LO, che capisce programmi scritti nel linguaggio L
    * Una implementazione di macchine astratte indipendente dalla macchina fisica
17. **Il costrutto for dei linguaggi C, C++ e Java non è un costrutto di iterazione determinata perché** 
    * L’esistenza di costrutti di iterazione derminata implicherebbe che C, C++ e Java non sono Turing-completi
    * Nessuna delle altre risposte
    * **Dall’interno del ciclo è possibile modificare il valore del contatore**
    * Non esistono costrutti di iterazione determinata
    * C, C++ e Java sono linguaggi imperativi
18. **Un oggetto denotabile (intendendo per “oggetto” una generica entità che può essere una variabile, una funzione, etc.…) è**
    * Un “oggetto” che può essere memorizzato in una variabile
    * Un “oggetto” per cui compare un binding nell’ambiente
    * **Nessuna delle altre risposte**
    * Un “oggetto” che ancora non compare nell’ambiente
    * Un “oggetto” che può essere generato come risultato da un’espressione complessa o da una funzione
19. **L’ambiente non locale di un blocco di codice è**
    * Nessuna delle altre risposte
    * L’insieme dei valori che le variabili non locali possono assumere
    * L’insieme dei binding creati all’interno del blocco di codice
    * **L’insieme dei binding visibili dentro al blocco, ma non direttamente definiti in esso**
    * Il subset dell’ambiente non visibile dentro al blocco di codice
20. **I puntatori di catena dinamica contenuti in un record di attivazione**
    * **Nessuna delle altre risposte**
    * Non esistono "puntatori di catena dinamica" in un record di attivazione
    * Servono per identificare la zona di memoria in cui è memorizzata una variabile locale
    * Devono essere esplicitamente allocati e deallocati dal codice del programma che li usa
    * Servono per accedere alle variabili dinamiche
21. **L’allocazione dinamica della memoria**
    * Nessuna delle altre risposte
    * È sempre effettuata solo dal compilatore o dall’interprete
    * Può essere fatta solo dallo heap
    * Può essere fatta solo dallo stack
    * **Può essere fatta sia dallo stack che dallo heap**
22. **Un compilatore da un linguaggio L ad un linguaggio LO è**
    * **Nessuna delle altre risposte**
    * Una implementazione di macchine astratte indipendente dalla macchina fisica
    * Un programma che trasforma un programma PLO (espresso nel linguaggio LO) in un programma PL (espresso nel linguaggio L) tale che per ogni input I si ha PL(I) = PLO(I)
    * L'implementazione di una macchina astratta scritta nel linguaggio LO, che capisce programmi scritti nel linguaggio L
    * Un programma scritto nel linguaggio LO che riceve come ingresso un programma PL (espresso nel linguaggio L) ed il suo input I generando lo stesso output che genera PL con input I
23. **I dangling pointer**
    * Sono identificabili tramite tecniche di reference counting (contatore dei riferimenti)
    * Sono un problema solo per il linguaggio Java
    * **Sono identificabili tramite la tecnica detta "mark and sweep"**
    * Non possono essere identificati con certezza, ma questo non è un problema perché comportano solo un piccolo spreco di memoria
    * Nessuna delle altre risposte
24. **La frammentazione esterna causa**
    * Uno spreco di memoria
    * Nessuna delle altre risposte
    * **L'impossibilità di allocare grandi blocchi di memoria anche se la memoria libera totale è sufficiente**
    * Un rallentamento rilevante nelle operazioni di allocazione della memoria
    * Il funzionamento non corretto di programmi che allocano memoria dinamicamente
25. **La valutazione con corto circuito del predicato "A && B" (dove "&&" rappresenta un "AND" logico)**
    * Stabilisce che se "B" è vero allora "A" non viene valutato
    * Nessuna delle altre risposte
    * **Stabilisce che se "A" è falso allora "B" non viene valutato**
    * Crea un non-determinismo nell'ordine della valutazione di "A" e "B"
    * Stabilisce che "A" e "B" devono essere valutati in parallelo
26. **Se l'ambiente di una funzione non contiene il nome della funzione stessa**
    * Nessuna delle altre risposte
    * **Non è possibile per la funzione invocarsi ricorsivamente**
    * Non ci sono particolari conseguenze
    * La funzione non può usare scope dinamico
    * La funzione non può usare scope statico
27. **La frammentazione interna causa**
    * Il funzionamento non corretto di programmi che allocano memoria dinamicamente
    * Nessuna delle altre risposte
    * Un rallentamento rilevante nelle operazioni di allocazione della memoria
    * **Uno spreco di memoria**
    * L'impossibilità di allocare grandi blocchi di memoria anche se la memoria libera totale è sufficiente
28. **L'allocazione dinamica della memoria**
    * **Può essere fatta sia dallo stack che dallo heap**
    * Nessuna delle altre risposte
    * È sempre effettuata solo dal compilatore o dall'interprete
    * Può essere fatta solo dallo stack
    * Può essere fatta solo dallo heap
29. **L'ambiente (o environment) è**
    * **L'insieme delle associazioni (nome, entità denotabile) esistenti in uno specifico punto del programma ed in uno specifico momento durante l'esecuzione di un programma**
    * L'insieme dei valori che una variabile assume durante l'esecuzione di un programma
    * Un insieme di associazioni (nome, valore) definite staticamente durante lo sviluppo di un programma
    * Nessuna delle altre risposte
    * Una lista di coppie (nome, tipo) che permette di accedere alle variabili di un programma
30. **Si può dire che una macchina astratta che capisce il linguaggio Java non sia implementata in modo puramente compilativo perché**
    * **Non esistendo un vero e proprio runtime per Java, non si può parlare di compilazione pura**
    * La macchina virtuale di Java (JVM) deve comunque essere compilata
    * Una macchina astratta che capisca un linguaggio di alto livello come Java non è mai implementabile con un compilatore
    * Non esistono compilatori Java
    * Nessuna delle altre risposte
31. **La ricorsione in coda**
    * Permette di risolvere il problema della ricorsione infinita
    * Richiede di non scrivere mai la chiamata ricorsiva come ultimo statement di una subroutine
    * Richiede di non ritornare mai direttamente il valore ritornato da una chiamata ricorsiva
    * Nessuna delle altre risposte
    * **Permette di evitare un'eccessiva crescita della dimensione dello stack**
32. **La memoria gestita staticamente**
    * È allocata esplicitamente dal programma a tempo di esecuzione, ma una volta allocata è staticamente legata al programma e non può essere liberata fino alla sua terminazione
    * È allocata prima dell’esecuzione del programma. Le entità allocate staticamente possono essere deallocate durante l’esecuzione del programma, per liberare memoria
    * **È allocata dal compilatore prima dell’esecuzione del programma. Le entità allocate staticamente in memoria risiedono in una zona fissa di memoria durante tutta l’esecuzione del programma**
    * È una memoria a sola lettura Nessuna delle altre risposte
33. **Un interprete di un linguaggio L scritto in un linguaggio LO è**
    * Nessuna delle altre risposte
    * L'implementazione di una macchina astratta scritta nel linguaggio LO, che capisce programmi scritti nel linguaggio L
    * Una implementazione di macchine astratte indipendente dalla macchina fisica
    * **Un programma scritto nel linguaggio LO che riceve come ingresso un programma PL (espresso nel linguaggio L) ed il suo input I generando lo stesso output che genera PL con input I**
    * Un programma che trasforma un programma PL (espresso nel linguaggio L) in un programma PLO (espresso nel linguaggio LO) tale che per ogni input I si ha PL(I) = PLO(I)
34. **In caso di scope statico**
    * Non è possibile annidare più blocchi di istruzioni
    * I legami fra nomi ed oggetto possono essere determinati solo a tempo di esecuzione
    * Il valore assegnato ad una variabile non può essere modificato
    * **I legami fra nomi ed oggetto possono essere determinati semplicemente leggendo il testo di un programma**
    * Nessuna delle altre risposte
35. **Un garbage collector**
    * **Può essere implementato tramite la tecnica detta “mark and sweep”, che riesce sempre a identificare tutta la memoria allocata dinamicamente ma non più utilizzata**
    * Richiede un’implementazione complessa, usando la tecnica dei tombstone (pietre tombali)
    * È implementabile solo in linguaggi di programmazione funzionali
    * È implementabile tramite la tecnica di lucchetti e chiavi, che però può causare dei memory leak
    * Nessuna delle altre risposte
36. **Un’entità denotabile è**
    * Nessuna delle altre risposte
    * Un’entità che può essere generata come risultato da un’espressione complessa o da una Funzione
    * **Una generica entità a cui può essere dato un nome**
    * Un'entità che ancora non compare nell'ambiente
    * Un'entità che può essere memorizzata
37. **Un interprete di un linguaggio L scritto in un linguaggio LO è**
    * Un programma che trasforma un programma PL (espresso nel linguaggio L) in un programma PLO (espresso nel linguaggio LO) tale che per ogni input I si ha PL(I) = PLO(I)
    * **Nessuna delle altre risposte**
    * Una implementazione di macchine astratte indipendente dalla macchina fisica
    * Un programma scritto nel linguaggio L che dato un input I produce lo stesso output generato dallo stesso programma scritto nel linguaggio LO
    * L’implementazione di una macchina astratta scritta nel linguaggio LO, che capisce programmi scritti nel linguaggio L
38. **L’ambiente (o environment) è**
    * **Nessuna delle altre risposte**
    * Un insieme di associazioni (nome, valore) definite staticamente durante lo sviluppo di un programma
    * L’insieme delle associazioni (variabile, valore) esistenti in uno specifico punto del programma ed in uno specifico momento durante l’esecuzione di un programma
    * L’insieme dei valori che una variabile assume durante l’esecuzione di un programma
    * Una lista di coppie (nome, tipo) che permette di accedere alle variabili di un programma
39. **Un garbage collector**
    * Richiede un’implementazione complessa, usando la tecnica dei tombstone (pietre tombali)
    * È implementabile solo in linguaggi di programmazione funzionali
    * Può essere implementato tramite la tecnica del reference counting (contatore dei riferimenti), che riesce sempre ad identificare tutta la memoria allocata dinamicamente ma non più utilizzata
    * È implementabile tramite la tecnica di lucchetti e chiavi, che però può causare dei memory leak
    * **Nessuna delle altre risposte**
40. **La memoria gestita dinamicamente**
    * È usata solo in linguaggi interpretati
    * Non è mai strettamente necessaria, ma permette di ottenere migliori prestazioni
    * **È necessaria per implementare la ricorsione**
    * Nessuna delle altre risposte
    * È necessaria per implementare l'iterazione
41. **L’ambiente (o environment) è**
    * Un insieme di associazioni (nome, valore) definite staticamente durante lo sviluppo di un programma
    * Una lista di coppie (nome, tipo) che permette di accedere alle variabili di un programma
    * Nessuna delle altre risposte
    * L’insieme dei valori che una variabile assume durante l’esecuzione di un programma
    * **L’insieme delle associazioni (nome, oggetto denotabile) esistenti in uno specifico punto del programma ed in uno specifico momento durante l’esecuzione di un programma**
42. **In caso di scope dinamico**
    * Il valore assegnato ad una variabile non può essere modificato
    * Non è possibile annidare più blocchi di istruzioni
    * Nessuna delle altre risposte
    * **I legami fra nomi ed oggetto possono essere determinati solo a tempo di esecuzione**
    * I legami fra nomi ed oggetto possono essere determinati semplicemente leggendo il testo di un programma
43. **I puntatori di catena statica contenuti in un record di attivazione**
    * Nessuna delle altre risposte
    * Servono per accedere alle variabili statiche
    * Devono essere esplicitamente allocati e deallocati dal codice del programma che li usa
    * **Servono per identificare la zona di memoria in cui è memorizzata una variabile in caso di scope statico**
    * Non esistono puntatori di catena statica in un record d'attivazione
44. **Un’entità denotabile può avere un tempo di vita più lungo di quello delle associazioni (fra l'entità e identificatori) che lo riferiscono**
    * Nessuna delle altre risposte
    * **Se l'entità è allocata dinamicamente dallo heap in una subroutine**
    * Mai
    * Se l'entità è allocata dinamicamente dallo stack in una subroutine
    * Se si usa scope dinamico
45. **La memoria gestita dinamicamente**
    * Non è mai strettamente necessaria, ma permette di ottenere migliori prestazioni
    * È necessaria per implementare l'iterazione
    * È usata solo in linguaggi interpretati
    * È necessaria solo per implementare macchine astratte per linguaggi compilati
    * **Nessuna delle altre risposte**
46. **I record d'attivazione**
    * Devono essere esplicitamente allocati e deallocati dal codice del programma che li usa
    * **Possono essere allocati sia sullo stack sia sullo heap (in caso, per esempio, di funzioni di ordine superiore)**
    * Sono necessari solo in presenza di funzioni di ordine superiore
    * Sono allocati solo sullo heap
    * Nessuna delle altre risposte
47. **Quale delle seguenti affermazioni riguardanti il garbage collector è vera**
    * Per Garbage Collection si intende una modalità automatica di gestione della memoria
    * L’utente alloca liberamente memoria
    * Non è permesso deallocare memoria
    * Il sistema periodicamente recupera la memoria allocata e non più utilizzabile
    * Un sistema operativo, o un compilatore, e un modulo di run-time liberano porzioni di memoria non più utilizzate dalle applicazioni.
    * **Tutte sono corrette**
48. **Quale delle seguenti affermazioni riguardanti il Garbage Collector è falsa**
    * Il garbage collector annoterà le aree di memoria non più referenziate e le libererà automaticamente
    * L’utente alloca liberamente memoria
    * **È permesso deallocare memoria**
    * Un sistema operativo, o un compilatore, e un modulo di run-time liberano porzioni di memoria non più utilizzate dalle applicazioni.
    * Il sistema periodicamente recupera la memoria allocata e non più utilizzabile
49. **Considerando la macchina astratta che permette di eseguire un linguaggio L, il numero di record di attivazione contemporaneamente presenti nel sistema:**
    * È sempre determinabile a priori da un’analisi statica del codice del programma eseguito
    * **Nessuna delle altre risposte**
    * Ha un massimo noto a priori solo se la macchina astratta alloca i record di attivazione dallo heap
    * Può avere un massimo noto a priori solo se il linguaggio L permette ricorsione
    * Non dipende dal programma che si esegue
50. **L’utilizzo della ricorsione in coda (tail recursion) in una funzione:** 
    * Utilizza più memoria
    * Nessuna delle altre risposte
    * **Riduce la possibilità di avere stack overflow**
    * Riduce l’efficienza della funzione
    * Richiede rimozione manuale degli stack frame
51. **La chiusura (closure) è:**
    * Nessuna delle altre risposte
    * espressione, ambiente (expression, environment) dove quest’ultimo ha almeno tutte le variabili nell’espressione unitamente all’indirizzo di ritorno
    * espressione, ambiente (expression, environment) dove quest’ultimo ha alcune delle variabili nell’espressione
    * **espressione, ambiente (expression, environment) dove quest’ultimo ha almeno tutte le variabili nell’espressione**
    * utilizzata per le call by reference
52. **Polish notation. Indicare il risultato di x – 5 6 7:**
    * -37
    * -5
    * 23
    * **-7**
    * Nessuna delle altre risposte
53. **Un linguaggio di programmazione (come FORTRAN) usa un'allocazione fissa della memoria (at compile time). Questo tipo di linguaggio non può supportare:**
    * floating point numbers
    * Exceptions
    * **variable size arrays**
    * nessuna delle precedenti
    * Procedures
54. **La differenza fra espressione e comandi:**
    * Un comando può modificare l'ambiente (locale, non locale o globale) mentre una espressione no
    * Non esistono differenze: sono due nomi diversi per la stessa entità sintattica
    * **Nessuna delle altre**
    * L'esecuzione di un comando produce un valore (o non termina), mentre la valutazione di una espressione può non produrre valori
    * Le espressioni non possono usare simboli appartenenti ad ambienti non locali
55. **Point reversal viene usato per garbage collection:**
    * per evitare di sprecare spazio
    * nessuna delle precedenti
    * nelle implementazioni di python
    * **quando non si ha spazio per una search structure**
    * per gestire loops nelle strutture puntatore
56. **Il cut=! i Prolog:**
    * **Ha successo e ferma il backtracking**
    * nessuna delle altre
    * esegue il backtracking
    * Chiama una eccezione
    * fallisce e ferma il backtracking
57. **Call by name**
    * Usata raramente al giorno d'oggi
    * Utilizzata in molti linguaggi procedurali
    * Richiede il passaggio dei valori delle variabili sullo stack
    * Ha una definizione semplice
    * **Nessuna delle altre risposte**
58. **Un'espressione può essere beta-ridotta ad un’unica forma normale**
    * Mai
    * Sempre
    * **Solo se la riduzione termina**
    * Ci possono essere diverse forme normali
    * Nessuna delle altre risposte
59. **L'iterazione determinata**
    * Non è implementabile in nessun linguaggio
    * Permette di modificare la variabile di controllo durante l'esecuzione del corpo dell'iterazione
    * Non permette di sapere in anticipo quante volte si ripeterà indipendentemente dal corpo del ciclo
    * è presente in C, C++ e Java
    * **Nessuna delle altre risposte**
60. **TextIO.lookahead**
    * **Legge il carattere successivo ma lo lascia nell'input stream**
    * Verifica se c'è un altro carattere nell'input stream
    * error
    * Legge il carattere successivo
    * Nessuna delle altre risposte

**Domande lambda calcolo**

1. **Beta-riducendo (λn.λm.λf.λx.(nf)((mf)x))(λf.λx.ffffx)(λf.λx.fx) si ottiene**
   * λf.λx.ffffx
   * f
   * Nessuna delle altre risposte
   * λf.λx.ffffffx
   * x
   * **λf.λx.fffffx**
2. **Beta-riducendo (λn.λm.λf.λx.(nf)((mf)x))(λf.λx.fx)(λf.λx.x) si ottiene**
   * λf.λx.f(f(f(fx)))
   * **λf.λx.fx**
   * x
   * Nessuna delle altre risposte
   * Fx
3. **Beta-riducendo (λn.λf.λx.f((nf)x))(λf.λx.f(f(f(fx)))) si ottiene**
   * La riduzione non termina
   * fx
   * **λf.λx.fffffx**
   * Nessuna delle altre risposte
   * λf.λx.f(f(f(fx))))
4. **Beta-riducendo (λa.((aλb.λc.c)λd.λe.d))(λf.λg.f) si ottiene**
   * **λb.λc.c**
   * Nessuna delle altre risposte
   * La riduzione non termina
   * c
   * λb.λc.b
5. **Beta-riducendo (λx.xy)(λz.zx)(λz.zx) si ottiene**
   * Nessuna delle altre risposte
   * Xyz
   * (λx.xy)yx
   * **yx(λz.zx)**
   * La riduzione non termina
6. **Beta-riducendo (λn.λf.λx.f((nf)x))(λf.λx.ffffx) si ottiene**
   * Nessuna delle altre risposte
   * λf.λx.ffffffx
   * La riduzione non termina
   * **λf.λx.fffffx**
   * fx
   * L’espressione è irriducibile
7. **Beta f-riducendo ((λa.aaa)(λb.b))(λc.c) si ottiene**
   * La riduzione non termina
   * Aaa
   * λx.xa
   * Nessuna delle altre risposte
   * **λc.c**
8. **Beta f-riducendo (λa.aaa)((λb.b)(λc.c)) si ottiene**
   * **λa.a**
   * Nessuna delle altre risposte
   * Aaa
   * La riduzione non termina
   * λx.xa
9. **Beta f-riducendo (λd.((λa.abc)(bc)))(λy.xyz) si ottiene**
   * La riduzione non termina
   * Nessuna delle altre risposte
   * λb.bba
   * λa.abc
   * **bcbc**
10. **Beta f-riducendo (λn.λf.λx.f((nf)x))(λf.λx.f(f(f(fx)))) si ottiene**
    * Nessuna delle altre risposte
    * L’espressione è irriducibile
    * La riduzione non termina
    * λf.λx.fffffx
    * fx
    * **λf.λx.f(f(f(f(fx))))**
11. **Beta-riducendo (λn.λm.λf.λx.(nf)((mf)x))(λf.λx.fx)(λf.λx.x) si ottiene**
    * Nessuna delle altre risposte
    * x
    * λf.λx.ffffffx
    * **λf.λx.fx**
    * fx
    * λf.λx.ffffx
12. **Beta-riducendo (λa.((aλb.λc.c)λd.λe.d))(λf.λg.g) si ottiene**
    * Nessuna delle altre risposte
    * **λb.λc.b**
    * b
    * λb.λc.c
    * La riduzione non termina
13. **Beta-riducendo (λa.aab)((λa.aab)(λa.(λb.ba)c)) si ottiene**
    * La riduzione non termina
    * aab
    * λa.aab
    * ccb(ccb)c
    * **Nessuna delle altre risposte**
14. **Beta riducendo (λa.aab)((λa.aab)(λa.(λb.ba)c)) si ottiene**
    * **Nessuna delle altre risposte**
    * ccb(ccb)c
    * λa.aab
    * La riduzione non termina
    * Aab
15. **Beta-riducendo ((λa.aaa)(λb.b))(λc.c) si ottiene**
    * **λc.c**
    * La riduzione non termina
    * Aaa
    * λx.xa
    * Nessuna delle altre risposte
16. **Beta-riducendo (λd.((λa.abc)(bc)))(λy.xyz) si ottiene**
    * La riduzione non termina
    * **bcbc**
    * Nessuna delle altre risposte
    * λa.abc
    * Λb.bb

# **Domande calcolo indirizzo array**

1. **Se gli array sono memorizzati per righe e int a[100][100] è un array multidimensionale di interi (si assuma che la dimensione di un intero sia 4 byte) con a[0][0] che ha indirizzo 0x5000, qual è l'indirizzo di a [5][10]?**
   * **0x57F8**
   * Nessuna delle altre risposte
   * 0x53ED
   * 0x5510
   * 0x51FE
2. **Se gli array sono memorizzati per colonne ed int a[100][100] è un array multidimensionale di interi (si assuma che la dimensione di un intero sia 4 byte)con a[0][0] che ha indirizzo 0x5000, qual è l'indirizzo di a[5][10]?**
   * 0x5510
   * 0x53ED
   * **Nessuna delle altre risposte**
   * 0x500F
   * 0x41FE
3. **Se gli array sono memorizzati per righe e char a[100][100][100] è un un array multidimensionale di caratteri con a[0][0] che ha indirizzo 0x1000, qual è l’indirizzo di a[5][5][10]?**
   * Nessuna delle altre risposte
   * 0x51510
   * 0xC54E
   * **0xD54E**
   * 0x50510
4. **Se gli array sono memorizzati per colonne e char a[100][100][100] è un un array multidimensionale di caratteri con a[0][0] che ha indirizzo 0x1000, qual è l’indirizzo di a[5][5][10]?**
   * Nessuna delle altre risposte
   * 0x18899
   * 0x51510
   * 0xD54E
   * **0x19899**
5. **Se gli array sono memorizzati per colonne e int a[25][25] è un un array multidimensionale di interi (si assuma che un intero sia memorizzato in 4 byte) con a[0][0] che ha indirizzo 0x1000, qual è l’indirizzo di a[5][10]?**
   * 0x11FE
   * Nessuna delle altre risposte
   * 0x100F
   * **0x13FC**
   * 0x121C
6. **Se gli array sono memorizzati per colonne ed short int a[100][100] è un array multidimensionale di interi corti (si assuma che la dimensione di uno short int sia 2 byte) con a[0][0] che ha indirizzo 0x4100, qual è l'indirizzo di a[5][10]?**
   * **0x48DA**
   * 0x4510
   * Nessuna delle altre risposte
   * 0x47DA
   * 0x41FE
7. **Se gli array sono memorizzati per colonne e char a[100][100] è un array multidimensionale di caratteri con a[0][0] che ha indirizzo 0x1100, qual è l'indirizzo di a[5][10]?**
   * Nessuna delle altre risposte
   * 0x24ED
   * 0x21FE
   * 0x22FE
   * **0x14ED**
8. **Se gli array sono memorizzati per colonne e char a[100][100] è un array multidimensionale di caratteri con a[0][0] che ha indirizzo 5243, qual è l'indirizzo di a[5][10]? (i numeri sono in decimale)**
   * Nessuna delle altre risposte
   * 5753
   * **6248**
   * 4730
   * 4955
9. **Se gli array sono memorizzati per colonne ed short int a[100][100] è un array multidimensionale di interi corti (si assuma che la dimensione di uno short int sia 2 byte) con a[0][0] che ha indirizzo 0x4100, qual è l’indirizzo di a[5][10]?**
   * 0x41FE
   * 0x500F
   * 0x47DA
   * 0x4510
   * 0x43ED
   * **Nessuna delle altre risposte**
10. **Se gli array sono memorizzati per righe ed int a[100][100] è un un array multidimensionale di interi (si assuma che la dimensione di un intero sia 4 byte) con a[0][0] che ha indirizzo 0x1000, qual è l’indirizzo di a[5][10]?**
    * 0x100F
    * **Nessuna delle altre risposte**
    * 0x13ED
    * 0x11FE
    * 0x1510
11. **Se gli array sono memorizzati per righe e char a[100][100] è un array multidimensionale di caratteri con a[0][0] che ha indirizzo 0x2000, qual è l’indirizzo di a[5][10]?**
    * 0x200F
    * **0x21FE**
    * Nessuna delle altre risposte
    * 0x23ED
    * 0x2510
12. **Se gli array sono memorizzati per colonne ed int a[100][100] è un array multidimensionale di interi (si assuma che la dimensione di un intero sia 4 byte) con a[0][0] che ha indirizzo 0x5000, qual è l'indirizzo di a[5][10]?**
    * **Nessuna delle altre risposte**
    * 0x53ED
    * 0x500F
    * 0x41FE
    * 0x551
13. **Se gli array sono memorizzati per righe e char a[100][100] è un array multidimensionale di caratteri con a[0][0] che ha indirizzo 0x1100, qual è l’indirizzo di a[5][10]?**
    * 0x22FE
    * **0x12FE**
    * 0x1510
    * 0x210F
    * Nessuna delle altre risposte
14. **Se gli array sono memorizzati per colonne ed short int a[100][100] è un array multidimensionale di interi corti (si assuma che la dimensione di uno short int sia 2 byte) con a[0][0] che ha indirizzo 16640, qual è l’indirizzo di a[5][10]?:**
    * **18650**
    * 18394
    * 17680
    * 16894
    * Nessuna delle altre risposte

# **Domande tipi funzioni ML**

1. **Qual è il tipo della seguente espressione:**

fun f x y = x y;

* + fun: 'a->'b
  + fun: 'a->'b->string->string
  + **fun: ('a->'b)->'a->'b**
  + fun: int->int
  + fun: string->string->string->string

1. **Qual è il tipo della seguente espressione:**

List.map print string;

* + fun : string list -> string list
  + **fun : string list -> unit list**
  + fun : string list -> string
  + fun : string -> string list
  + fun: string -> unit
  + fun: 'a list -> 'b list

1. **Qual è il tipo della seguente espressione:**

let f1 l = List.map (fun x y -> x,y) l;

* + f1:'a list->('b -> 'a -> 'b) list
  + f1:'a list-> 'b -> ('a \* 'b)
  + fun x y -> x,y :'a -> 'b -> 'a \* 'b
  + fun x y -> x,y :'a -> 'b -> 'a -> 'b
  + **f1:'a list->('b -> 'a \* 'b) list**

1. **Qual è il tipo della seguente espressione:**

(fn x => x) 12;

* + **12: int**
  + 'a: 12
  + string: "12"
  + fn: 'a -> 'a
  + Nessuna delle altre risposte

1. **Qual è il tipo della seguente espressione:**

fun f1 x = [x,x];

* + f1 = fn: 'a list -> 'a list
  + f1 = fn: 'a list -> 'b list
  + f1 = fn: 'a -> 'b
  + **f1 = fn: 'a -> 'a list**
  + f1 = fn: 'a -> 'a
  + f1 = fn: string -> string

1. **Qual è il tipo della seguente espressione:**

fun f2 x y = (x @ (y x));

* + fun: 'a -> 'a -> 'a -> 'a
  + **fun: 'a list -> ('a list -> 'a list) -> 'a list**
  + fun: 'a list -> 'a list -> 'a list
  + fun: 'a list -> 'a list -> 'b list
  + fun: 'a -> ('a list -> 'a list) -> 'a list

1. **Qual è il tipo della seguente espressione:**

fun f3 x = List.map x;

* + fun : ('a list -> 'b list) -> 'a list -> 'b list
  + fun : ('a -> 'b) -> 'a -> 'b
  + fun : ('a -> 'a) -> 'a list -> 'a list
  + fun : 'a list -> 'b list
  + **fun : ('a -> 'b) -> 'a list -> 'b list**

1. **Qual è il tipo della seguente espressione:**

fn x => fn y => x+y;

* + fn: int -> int
  + fn: 'a -> 'a -> 'a
  + **fn: int -> int -> int**
  + fn: 'a -> 'a
  + fn: int -> int -> 'a
  + Nessuna delle altre risposte

1. **Qual è il tipo della seguente espressione:**

fun funzione x y z = x ^ y ^ z

* + fn: string -> string
  + fn: 'a -> 'a
  + fn: 'a -> 'a -> 'a -> 'a
  + fn: string -> string -> string -> int
  + Nessuna delle altre risposte
  + **fn: string -> string -> string -> string**

1. **Qual è il tipo della seguente espressione:**

fun c a1 a2 = a1::a2;

* + fn: 'a list -> 'a list -> 'a list
  + **fn: 'a -> 'a list -> 'a list**
  + fn: 'a -> 'a -> 'a
  + fn: 'a -> 'a list
  + fn: 'a -> 'a -> 'a list

1. **Qual è il tipo della seguente espressione:**

fn l1 => fn l2 => l1@l2;

* + **fn: 'a list -> 'a list -> 'a list**
  + fn: 'a -> 'a list -> 'a list
  + fn: 'a -> 'a -> 'a
  + fn: 'a -> 'a list
  + fn: 'a -> 'a -> 'a list

1. **Qual è il tipo della seguente espressione:**

fn a => fn b => fn c => (a \* b) / c;

* + fn: int -> int -> int -> int
  + fn: 'a -> 'a -> 'a -> 'a
  + fn: int -> int -> int -> real
  + **fn: real -> real -> real -> real**
  + fn: int -> int -> real -> real

1. **Qual è il tipo della seguente espressione:**

fun f [] = (7.0; print("ab")) | f (x::xs) = (2.3; print("cd"));

* + **val f = fn: 'a list -> unit**
  + 'a list -> int
  + 'a tuple
  + val f = fn: 'a list -> real
  + real list -> string

1. **Qual è l'output del seguente codice:**

List.partition(fn x => x > 0) (List.take ([~1,1,~2,~2,~3,3],4));

* + val it = ([1,3], [~1, ~2, ~2, ~3]): int list \* int list
  + val it = [1,3]: int list
  + val it = [1]: int list
  + **val it = ([1], [~1, ~2, ~2]): int list \* int list**
  + val it = ([], [~2]): int list \* int list
  + val it = []: int list

1. **Qual è il tipo della seguente espressione:**

fn a => fn b => a = b;

* + **fn: ''a -> ''a -> bool**
  + fn: int -> int -> bool
  + fn: string -> string -> bool
  + fn: bool -> bool -> bool
  + fn: 'a -> 'a -> bool

1. **Qual è il type di:**

fun f[] = (print("ab"); 7.0) | f(x::xs) = (print("cd"); 4.0);

* + ‘a list --> int
  + “a --> real
  + ‘a record
  + ‘a tuple
  + **‘a list --> real**

1. **In ML ":>" serve a**
   * nessuna delle altre risposte
   * nasconde i componenti di una lista
   * è usata per definire funzioni polimorfiche
   * **nasconde i componenti di una struttura**
   * **definisce la firma di una struttura**
2. **Dato il seguente programma in ML, indicare il risultato:** Datatype fruit = Apple | Pear | Grape; isApple x = (x=Apple); isApple (Lemon);
   * nessuna delle altre risposte
   * False
   * **undeclared value**
   * True
   * type error
3. **Qual è l'output del seguente codice:** val i=ref 1; while !i<10 do i := !i+1;
   * val i=10
   * **val it=()**
   * Loop infinito
   * Type error
   * Nessuna delle altre risposte
4. **Qual è l'output del seguente codice: val m = (List.map(fn x=>x\*x [4, 5, 6]; List.foldr Int.+0 m;**
   * 15
   * **77**
   * 'a list -> int'
   * NaN
   * Nessuna delle altre risposte

# **Domande output codici**

|  |  |
| --- | --- |
| p = malloc();  q = malloc();  \*p = 999  \*q = 111  \*p += \*q  q = p free(p) | 1. **Utilizzando la tecnica dei tombstones cosa succede alla fine del frammento di codice dato?**    * \*p = 1110 e \*q = 111    * **p è deallocato e q punta ad una tombstone**    * p e q sono entrambi deallocati    * q è deallocato e p punta ad una tombstone    * \*p = 999 e \*q = 111    * \*p = 1110 e \*q = 1110 |
| p = malloc();  q = malloc();  \*p = 999  \*q = 111  \*p += \*q  q = p  free(p) | 1. **Utilizzando la tecnica dei tombstones cosa succede alla fine del frammento di codice dato?**    * \*p = 1110 e \*q = 111    * \*p = 1110 e \*q = 1110    * p e q sono entrambi deallocati    * q è deallocato e p punta ad una tombstone    * **Nessuna delle risposte**    * \*p = 999 e \*q = 111 |
| p = malloc();  q = malloc();  \*p = 123  \*q = 321  p = q  free(q) | 1. **Utilizzando la tecnica dei tombstones cosa succede alla fine del frammento di codice dato?**    * \*p = 123 e \*q = 321    * p è deallocato e q punta ad una tombstone    * p e q sono entrambi deallocati    * **q è deallocato e p punta ad una tombstone**    * \*p = 321 e \*q = 321    * Nessuna delle risposte |
| int c = 2;  int pippo(int a) {  c = c + 2;  return a \* 2;  }  int pluto(void) {  return pippo(c + 1);  } | 1. **Si consideri lo pseudo-codice. Qual è il valore di ritorno di pluto() se i parametri sono passati per nome?**    * Nessuna delle altre risposte    * **10**    * Dipende dal tipo di scope (statico o dinamico) utilizzato    * 4    * 6 |
| int c = 2;  int pippo(int a) {  c = c + 2;  return a \* 2;  }  int pluto(void) {  return pippo(c + 1);  } | 1. **Si consideri lo pseudo-codice. Qual è il valore di ritorno di pluto() se i parametri sono passati per valore?**    * 10    * Dipende dal tipo di scope (statico o dinamico) utilizzato    * Nessuna delle altre risposte    * **6**    * Non è possibile passare c + 1 per valore |
| int r(int x) {  return r(x - 1);  }  int f(int a, int b, int c) {  if (c == 1)  return a;  else  return b;  } | 1. **Si consideri lo pseudo-codice. Qual è il valore di ritorno di f(1,r(1),1) se i parametri sono passati per nome?**    * Non è possibile dirlo senza conoscere il tipo di scope (statico o dinamico) utilizzato    * Nessuna delle altre risposte    * **1**    * Si ha ricorsione infinita    * 0 |
| int a, b, c;  void pippo(void) {  int a;  a = 6;  b = 5;  }  void pluto(void) {  int c;  int b;  pippo();  c = 3;  a = 4;  }  void topolino(void) {  int a;  a = 1;  b = 10;  pluto();  c = a + b;  } | 1. **Dato il frammento di programma (espresso in pseudo-codice), quanto vale la variabile globale c dopo aver eseguito topolino(), assumendo scope dinamico?**    * **14**    * 3    * Non è possibile dirlo    * 6    * Nessuna delle altre risposte |
| int x, y, z;  void f3(void) {  x = 0;  y = 5;  }  void f2(void) {  int y;  f3();  y = 0;  z = 10;  }  int f1(void) {  int x;  x = -5;  y = 10;  z = x + y;  f2();  return z - y - x;  } | 1. **Dato il frammento di programma (espresso in pseudo-codice), qual è il valore di ritorno di f1(), assumendo scope statico?**    * 5    * **Nessuna delle altre risposte**    * Non è possibile dirlo    * 0    * -5 |
| int b = 666;  int pippo(int x) {  x = 666;  b = 1;  return x / 2;  }  int pluto(void) {  int a, c;  a = b / 333;  c = pippo(a);  return a + c;  } | 1. **Si consideri lo pseudo-codice. Qual è il valore di ritorno di pluto() se i parametri sono passati per valore?**    * **Nessuna delle altre risposte**    * 1    * 3    * 0    * Dipende dal tipo di scope (statico o dinamico) utilizzato |
| int somma(int a, int b) {  if (a == 0)  return b;  return somma(a-1, b+1); } | 1. **Funzione implementata dallo pseudo-codice**    * Non usa ricorsione in coda    * Non può essere implementata per via iterativa    * Nessuna delle altre risposte    * **Usa ricorsione in coda**    * Causa sempre ricorsione infinita |
| int f4(int x) {  return 1 / x;  }  int f3(int a, int b) {  if (a > 1)  return a;  else  return b;  } | 1. **Si consideri lo pseudo-codice. Qual è il valore di ritorno di f3(10,f4(0)) se i parametri sono passati per valore?**    * 10    * Non è possibile dirlo senza conoscere il tipo di scope (statico o dinamico) utilizzato    * Non è possibile passare f4(0) per valore    * **Nessuna delle altre risposte**    * 1 |
| int i = 0;  int x[10];  int f2(int z) {  i = i / 2;  return z - i;  }  int f1(void) {  int y;  i = 4;  x[0] = 1;  x[1] = 0;  x[2] = 2;  x[3] = 4;  x[4] = 6;  y = f2(x[i]);  return y + i;  } | 1. **Si consideri lo pseudo-codice. Qual è il valore di ritorno di f1() se i parametri sono passati per valore?**    * **Nessuna delle altre risposte**    * 0    * 2    * Dipende dal tipo di scope (statico o dinamico) utilizzato    * 4 |
| int x, y, z;  void f3(void) {  x = 0;  y = 5;  }  void f2(void) {  int y;  f3();  y = 0; z = 10;  }  int f1(void) {  int x;  x = -5;  y = 10;  z = x + y;  f2();  return z - y - x;  } | 1. **Dato il frammento di programma (espresso in pseudo-codice), qual è il valore di ritorno di f1(), assumendo scope dinamico?**    * Non è possibile dirlo    * 5    * **0**    * Nessuna delle altre risposte    * -5 |
| int b = 666;  int pippo(int x) {  x = 666;  b = 1;  return x / 2;  }  int pluto(void) {  int a, c;  a = b / 333;  c = pippo(a);  return a + c;  } | 1. **Si consideri lo pseudo-codice. Qual è il valore di ritorno di pluto() se i parametri sono passati per valore?**    * **Nessuna delle altre risposte**    * 666    * 999    * Dipende dal tipo di scope (statico o dinamico) utilizzato    * 333 |
| int somma(int a, int b) {  if (a == 0)  return b;  return somma(a-1, b)+1;  } | 1. **Funzione implementata dallo pseudo-codice**    * Nessuna delle altre risposte    * Non può essere implementata per via iterativa    * **Non usa ricorsione in coda**    * Causa sempre ricorsione infinita    * Usa ricorsione in coda |
| int b = 666;  int pippo(int x) {  x = 666;  b = 1;  return x / 2;  }  int pluto(void) {  int a, c;  a = b / 333;  c = pippo(a);  return a + c;  } | 1. **Si consideri lo pseudo-codice. Qual è il valore di ritorno di pluto() se i parametri sono passati per nome?**    * Non è possibile passare a per nome    * Dipende dal tipo di scope (statico o dinamico) utilizzato    * Nessuna delle altre risposte    * **999**    * 335 |
| int mistero(int a, int b) {  if (b == 0)  return a;  return mistero(a/2,b-1);  } | 1. **La funzione implementata dallo pseudo-codice**    * Può causare una crescita incontrollata dello stack    * Causa sempre ricorsione infinita    * **Nessuna delle altre risposte**    * Non può essere implementata per via iterativa    * Non usa ricorsione in coda |
| int x, y;  void pippo(void) {  x = 8;  y = 4;  }  void pluto(void) {  int y;  pippo();  y = 3;  }  int topolino(void) {  int x, z;  x = 5;  y = 15;  z = x + y;  pluto();  return z - y - x;  } | 1. **Dato il frammento di programma (espresso in pseudo-codice), qual è il valore di ritorno di topolino(), assumendo scope statico?**    * Nessuna delle altre risposte    * Non è possibile dirlo    * 0    * -3    * **11**    * -8 |
| int x, y, z;  void f3(void) {  x = 0;  y = 5;  }  void f2(void) {  int y;  f3();  y = 0;  z = 10;  }  int f1(void) {  int x;  x = -5;  y = 10;  z = x + y;  f2();  return z - y - x;  } | 1. **Dato il frammento di programma (espresso in pseudo-codice), qual è il valore di ritorno di f1(), assumendo scope dinamico?**    * **Nessuna delle altre risposte**    * -1    * -5    * 5    * Non è possibile dirlo |
| int c = 2;  int pippo(int a) {  c = c + 2;  return a \* 2;  }  int pluto(void) {  return(pippo(c + 1));  } | 1. **Si consideri lo pseudo-codice. Qual è il valore di ritorno di pluto() se i parametri sono passati per valore?**    * Nessuna delle altre risposte    * Dipende dal tipo di scope (statico o dinamico) utilizzato    * Non è possibile passare c + 1 per valore    * **6**    * 10 |
| int r(int x) {  return r(x - 1);  }  int f(int a, int b, int c) {  if (c == 1) return a;  else return b;  } | 1. **Si consideri lo pseudo-codice. Qual è il valore di ritorno di f(1,r(1),1) se i parametri sono passati per valore?**    * Nessuna delle altre risposte    * **Si ha ricorsione infinita**    * 1    * Non è possibile dirlo senza conoscere il tipo di scope (statico o dinamico) utilizzato    * 0 |
| int x, y, z;  void minni(void) {  x = 4;  y = 8;  }  void paperino(void) {  int y;  minni();  y = 1;  z = 666;  }  int topolino(void) {  int x;  x = 5;  y = 15;  z = x + y;  paperino();  return z - y - x;  } | 1. **Dato il frammento di programma (espresso in pseudo-codice), qual è il valore di ritorno di topolino(), assumendo scope statico?**    * -3    * 0    * 14    * **Nessuna delle altre risposte**    * Non è possibile dirlo |
| int c = 2;  int pippo(int a) {  c = c + 2;  return a \* 2;  }  int pluto(void) {  return(pippo(c + 1));  } | 1. **Si consideri lo pseudo-codice. Qual è il valore di ritorno di pluto() se i parametri sono passati per riferimento?**    * Dipende dal tipo di scope (statico o dinamico) utilizzato    * 10    * **Non è possibile passare c + 1 per riferimento**    * Nessuna delle altre risposte    * 6 |
| z=1;  for i=1 to 1+z by 1 do {  write(i);  z++;  }  write(z); | 1. **Usando la ”bounded iteration” (numerically controlled) indicare cosa viene stampato dal seguente codice**    * 1 2 3 4 ... (infinite loop)    * **1 2**    * 1 2 3    * Nessuna delle altre risposte    * 1 1 1 |
| int a, b, c;  void pippo(void) {  int a;  a = 6;  b = 5;  }  void pluto(void) {  int c;  int b;  pippo();  c = 3;  a = 4;  }  void topolino(void) {  int a;  a = 1;  b = 10;  pluto();  c = a + b;  } | 1. **Dato il frammento di programma (espresso in pseudo-codice), quanto vale la variabile globale c dopo aver eseguito topolino(), assumendo scope statico?**    * Nessuna delle altre risposte    * **6**    * Non è possibile dirlo    * 14    * 3 |
| int b = 666;  int pippo(int x) {  x = 666;  b = 1;  return x / 2;  }  int pluto(void) {  int a, c;  a = b / 333;  c = pippo(a);  return a + c;  } | 1. **Dato il frammento di programma (espresso in pseudo-codice) qual è il valore di ritorno di pluto() se i parametri sono passati per riferimento?**    * 333    * Nessuna delle altre risposte    * Non è possibile passare a per riferimento (perché a = b / 333)    * 666    * **999**    * 2 |
| p = malloc();  q = malloc();  \*p = 222;  \*q = 999;  \*q = \*p / 2;  q = p;  free(p); | 1. **Supponendo sia impiegata la tecnica delle “tombstone” per ovviare al problema dei dangling references, indicare quale delle seguenti affermazione è vera alla fine dell’esecuzione del codice:**    * q punta alla tombstone, p punta alla tombstone    * **Nessuna delle altre risposte**    * \*p=999, q punta alla tombstone    * p punta alla tombstone, \*q=111    * q risulta deallocato, \*p=222 |