1. **Che cosa sono e a che cosa servono le chiusure?(closure/thunk) In java è possibile passare come parametro un metodo ad un altro metodo? In C sono necessarie le chiusure?**
   Sono le coppie (espressione, ambiente) che servono per la corretta valutazione delle espressioni, ad esempio nei passaggi per nome o nelle funzioni di ordine superiore.
   Il linguaggio nativo non lo permette perché non è prevista la chiusura.
   In C non vengono usate le chiusure perché non esiste l'ambiente locale e per passare una funzione basta passare un puntatore al codice.
2. **Si mostri con un esempio la differenza tra scoping statico e scoping dinamico. Si scriva un programma che viene valutato in maniera diversa.**

```
{int i = 0;
void f()
        {print i}
void g()
        {int i = 3; f() }
g();
```

3. **Si descriva il passaggio per nome**
   Il parametro formale viene sostituito dal parametro attuale. Ci deve essere una sostituzione senza cattura di variabile e questo viene rispettato richiedendo che il parametro formale venga valutato nell'ambiente del chiamante e non in quello del chiamato. La semantica è stabilita nella regola di copia che consente di tenere il legame tra il parametro formale e quello attuale. L'implementazione è fornita da una chiusura.
4. **Metodo di passaggio dei parametri per riferimento. C ha il passaggio per riferimento?**
   Il parametro è sia di ingresso che di uscita. Al momento della chiamata viene valutato l'l-value del parametro attuale e l'ambiente locale è esteso con un'associazione tra il parametro formale e l'l-value dell'attuale. Al termine della procedura viene distrutto l'ambiente locale e il legame tra il formale e il legame tra l'l-value e l'attuale. Ogni modifica al formale è una modifica all'attuale perché viene creato un alias. No, ma in C questo sistema può essere simulato attraverso l'uso dei puntatori.
5. **Implementazione del passaggio dei parametri di tipo funzione.**
   È possibile passare le funzioni come parametri ad altre funzioni. Vi sono due possibilità per l'ambiente non locale da utilizzare al momento dell'esecuzione di una funzione f invocata tramite un parametro formale h
   a. utilizzare l'ambiente attivo al momento della creazione del legame tra h e f. Il linguaggio adotta una politica di *deep binding*
   b. utilizzare l'ambiente attivo al momento della chiamata di f tramite h. il linguaggio usa una politica di tipo *shallow binding*.

   Lo shallow binding ricerca per ogni nome la sua ultima associazione presente.
   Con deep binding l'informazione relativa al puntatore di catena statica deve essere determinata al momento dell'associazione tra il parametro formale e quello attuale. Al parametro formale viene associata una copia (testo ambiente) rappresentata a livello implementativo da una copia (puntatore al codice, puntatore al RDA). Questa è una chiusura. Al momento in cui il formale è usato per invocare una funzione la macchina astratta trova il codice a cui trasferire il controllo nella prima componente della chiusura e assegna il contenuto della seconda componente della chiusura al puntatore di catena statica del record di attivazione della nuova invocazione
6. **Si descriva il problema dell'upward funarg**
   Questo problema si presenta quando si ritorna una funzione può essere restituita come risultato. Quando una funzione chiama un'altra funzione durante l'esecuzione

di un programma, lo stato locale del chiamante (inclusi i parametri e variabili locali), deve essere preservato al fine di lasciar procedere l'esecuzione del chiamato dopo la terminazione del chiamante. Nella maggior parte dei programmi compilati lo stato locale è memorizzato nello stack (RDA). La soluzione sarebbe allocare gli RDA nello heap invece che nello stack e fare affidamento sul garbage collection oppure reference counting.

7. **Descrivere il metodo di passaggio dei parametri per value result e si mostri che tale metodo non è equivalente al passaggio per riferimento.**
   Il passaggio per value result realizza una comunicazione bidirezionale col parametro formale che è a tutti gli effetti una variabile locale. Al termine della procedura il valore corrente del parametro formale viene assegnato alla locazione corrispondente al parametro attuale. Con il passaggio per riferimento vengono creati degli alias alla stessa locazione di memoria.

```
void foo (reference/val result int x, reference/val result int y, reference int z)
   { y = 2; x =4; if (x==y) { z = 1 } }
….
int a = 3, b = 0; foo(a,a,b);
```

8. **Algoritmo di garbarbage collection mark and sweep**
   Ci sono due fasi fase di mark e quella di sweep. Nella fase di *mark* si attraversano una prima volta tutti gli oggetti dello heap marcando ciascuno come "inutilizzato". Partendo dai puntatori attivi sulla pila si attraversano tutte le strutture dati presenti sullo heap marcando come in "uso" ogni oggetto che viene attraversato. Nella fase di *sweep* i blocchi inutilizzati sono restituiti alla lista libera.
   Sono necessari dei descrittori che diano la dimensione di ogni blocco allocato. Questa tecnica viene invocata quando la memoria libera sullo heap è prossima ad esaurirsi.
   Difettil: crea frammentazione esterna; richiede un tempo proporzionale alla dimensione totale dello heap e diminuisce la località dei riferimenti.
9. **Che tipo di passaggio degli argomenti hanno i linguaggi ML,C e Java? In java cosa accade quando viene passato un array ad un metodo?**
   In C e Java per valore e costante
10.