CORSO DI LINGUAGGI DI PROGRAMMAZIONE
PROVA SCRITTA DEL 30 MAGGIO 2023 .

Tempo a disposizione: ore 2.
Svolgere gli esercizi 1–4 e 5–8 su due fogli differenti.

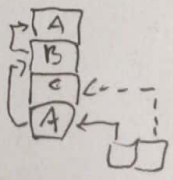
1. Si dica cosa stampa il seguente frammento in uno pseudolinguaggio con passaggio per nome e scope statico

```
int x = 2;
void foo(name int y){
    int z=y;
    z = y;
    x = z+y;
    write(x);
}
{int x = 10;
 foo(x++);
 write(x);
}
```

2. Il linguaggio imperativo Pippo permette di usare solo variabili di tipo intero e i seguenti comandi: assegnamento, if, case, for "vero" (iterazione determinata). Non ammette ricorsione e non ammette comandi di allocazione (e deallocazione) esplicita della memoria. Si dica, motivando la risposta, qual'è la più semplice forma di gestione della memoria utilizzabile nell'implementazione di Pippo e se è possibile determinare staticamente la dimensione massima della memoria richiesta da un programma scritto in Pippo.

3. Si consideri il seguente frammento di codice scritto in uno pseudo-linguaggio che usa scoping statico implementato mediante display.

```
{int x = 0;
 int A(reference int y) {
         int x =2;
         y=y+1;
         return B(y)+x;
         }
 int B(reference int y){
        int C(reference int y){
              int x = 3;
              return A(y)+x+y;
              }
        if (y==1) return C(x)+y;
        else return x+y;
        }
write (A(x));
}
```



Si dia graficamente la situazione del display e della pila dei record di attivazione al momento in cui il controllo entra per la *seconda* volta nella funzione A. Per ogni record di attivazione si dia solo il valore del campo destinato a salvare il valore precedente del display.

4. Si consideri il seguente frammento in uno pseudolinguaggio parametri di ordine superiore:

```
{void foo (int f(), int n){
    int m = 10;
    int fie(){
      write(n,m);
    }
    if (n==0) f();
    else {m = 30;
          foo(fie,0);
        }
 }
 int g(){
    write(10);
 }
 foo(g,1);
}
```

Si dica cosa stampa il frammento con scope dinamico e shallow binding.

5. In un linguaggio con passaggio per riferimento e supporto al polimorfismo di sottotipo e parametrico, sono dati i seguenti tipi per cui vale la relazione di sottotipaggio <: nelle seguenti direzioni: `Mammal <: Animal`, `Lion <: Carnivore <: Mammal` e `Giraffe <: Herbivore <: Mammal`. Viene inoltre definito il contenitore polimorfo `Cage[ T ]` dotato delle operazioni `add: T -> ()` e `remove: () -> T`, con `T` parametro di tipo. Il linguaggio offre l'istruzione `new` per creare una nuova istanza di un tipo e supporta sottotipi parametrici con la notazione `T[? <: S]` e `T[? :> S]` per indicare la relazione di sottotipaggio del tipo parametrico `?` rispetto ad un tipo concreto `S`.

   Nel codice sottostante, indicare quali istruzioni sono errate e spiegare brevemente perchè.

```
Cage[ ? :> Carnivore ] cage1 = new Cage[ Mammal ]();     // I1 ✓
Cage[ ? <: Mammal ]    cage2 = new Cage[ Carnivore ](); // I2 ✓
Cage[ ? <: Herbivore ] cage3 = new Cage[ Giraffe ]();   // I3 ✓
cage1.add( new Lion() );                                // I4 ✓?
Mammal a1 = cage2.remove();                             // I5 ✓
cage1.add( new Giraffe() );                             // I6 ✗
cage2.add( cage1.remove() );                            // I7 ✗
cage2.add( a1 );                                        // I8 ✗
Herbivore a2 = cage3.remove();                          // I9 ✓
cage2 = cage3;                                          // I10 ✓
cage3.add( a2 );                                        // I11 ✗
cage1.add( cage3.remove() );                            // I12 ✗
```

Stessa logica, sì o no?

6. Si assuma di avere uno linguaggio con passaggio per riferimento e con garbage collection mediante contatori dei riferimenti. Dato il seguente frammento di codice, indicare, insieme ad una breve spiegazione del ragionamento seguito, i valori dei contatori dei riferimenti dei tre oggetti al termine dell'esecuzione dell'ultima istruzione.

```
struct Elem {
 Elem next;
}

f(){
1 Elem a = new Elem(); // OGG1
2 Elem b = new Elem(); // OGG2
3 b.next = a;
4 a.next = b;
5 Elem c = new Elem(); // OGG3
6 c.next = a;
7 a.next.next = a.next;
8 c.next.next = a;
  return c;
}

Elem d = f()          2, 1, 1
```

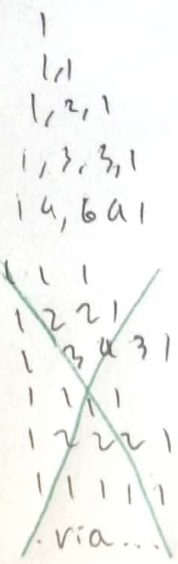
7. Si consideri un linguaggio nel quale le eccezioni vengono sollevate con l'istruzione `throw E` e vengono gestite coi blocchi `try{ ... } catch E { ... }`. Il linguaggio ha scoping statico per tutti i nomi, eccezioni comprese. Nel frammento sotto, si assuma che l'eccezione `MyException` sia visible a tutto il blocco. I valori `ListInt` si creano con l'istruzione `new`, includono nuovi elementi con l'operazione `add: int -> ()`, ottengono il numero di elementi contenuti con `size: () -> int` e ritornano un elemento ad una posizione specifica (di input) con `get: int -> int throws MyException`, dove, se l'elemento non esiste, `get` lancia l'eccezione `MyException`. Cosa stampa (tramite l'operazione `print`, che ad esempio, stampa una lista con gli elementi 1,2,3 come `[1,2,3]`) il seguente frammento? Spiegare brevemente il ragionamento dietro la risposta.

```
ListInt p( ListInt i, ListInt t ) throws MyException {
t.add( i.get( 0 ) );
try {
 for ( int j = 0; j < i.size(); j++ ) {
  t.add( i.get( j ) + i.get( j+1 ) );
 }
} catch ( MyException ){}
t.add( i.get( i.size() - 1 ) );
return t;
}

void g( ListInt i ) throws MyException {
 print( i );
 if( i.size() > 1 && i.size() < 5 ){
   try { g( p( i, new ListInt() ) ); }
   catch ( MyException ){}
 } else { throw MyException; }
}

f( ListInt ls ) throws MyException {
 ls.add( 1 );
 try { g( ls ); }
 catch ( MyException ){ f( ls ); }
}

f( new ListInt() )
```



8. Il seguente programma è scritto in un linguaggio ad oggetti con indizzamento dinamico dei metodi e statico delle variabili e dove `A extends B` significa che `B` è una sottoclasse di `A`. Indicare cosa stampa (tramite l'istruzione `print`) il programma e perchè (anche rappresentando quali meccanismi del linguaggio sono interessati).

```
class A {
 int x = 4;
 int m() { x=x+1; return x; }
}
class B extends A {
 int x = 6;
 int m() { x=x+2; return x; }
}
A x = new B();
print( x.x + x.m() );→12
       9   8
```