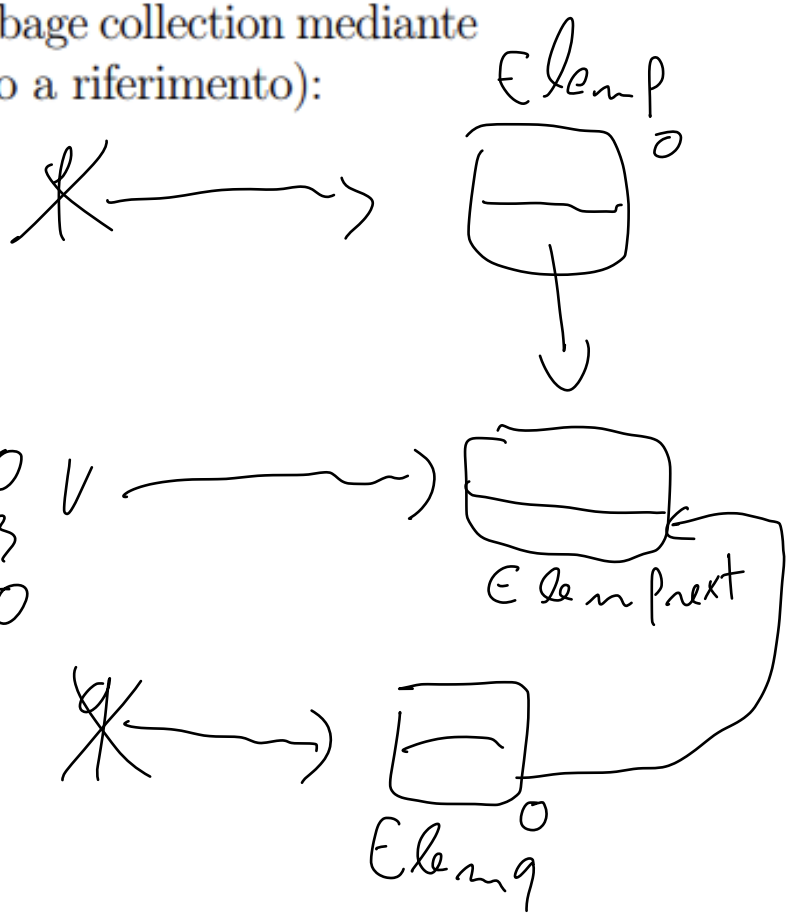
Quinta esercitazione
Linguaggi di programmazione

Tutor didattico: Giosuè Cotugno
giosue.cotugno2@unibo.it
A.A. 2023/2024

Esercizio 1

Si consideri il seguente frammento in un linguaggio con garbage collection mediante contatori dei riferimenti (le variabili adottano un modello a riferimento):

```
class Elem {
 int n;
 Elem next;

 Elem foo() {
  Elem p = new Elem();  // oggetto OGG1
  p.next = new Elem();  // oggetto OGG2
  Elem q = new Elem();  // oggetto OGG3
  q.next = p.next;
  return p.next;
 }
}
Elem r = Elem.foo();
```

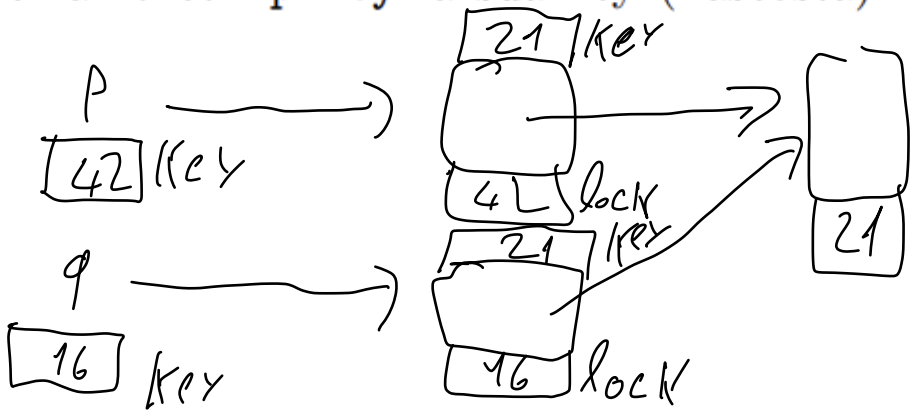


Si dica quali sono i valori dei contatori dei riferimenti dei tre oggetti al termine dell'esecuzione del frammento. Spiegare brevemente (max 10 righe) il ragionamento dietro alla risposta.

Esercizio 2

7. Si assuma di avere uno linguaggio che adotti la tecnica *locks and keys*. Dato `OGG` generico oggetto nello heap, indichiamo con `OGG.lock` il suo lock (nascosto). Data `p` variable contente il valore di un generico puntatore (sulla pila o nello heap), indichiamo con `p.key` la sua key (nascosta). Si consideri il seguente frammento di codice:

```
class C {
 C next;
}
C p = new C();          // oggetto OGG1
C q = new C();          // oggetto OGG2
{ p.next = new C(); } // oggetto OGG3
q.next = p.next;
```



Si diano possibili valori di `OGG1.lock`, `OGG2.lock`, `OGG3.lock`, `p.key`, `p.next.key`, `q.key`, `q.next.key` dopo l'esecuzione del frammento (spiegare brevemente il ragionamento seguito).

Esercizio 3

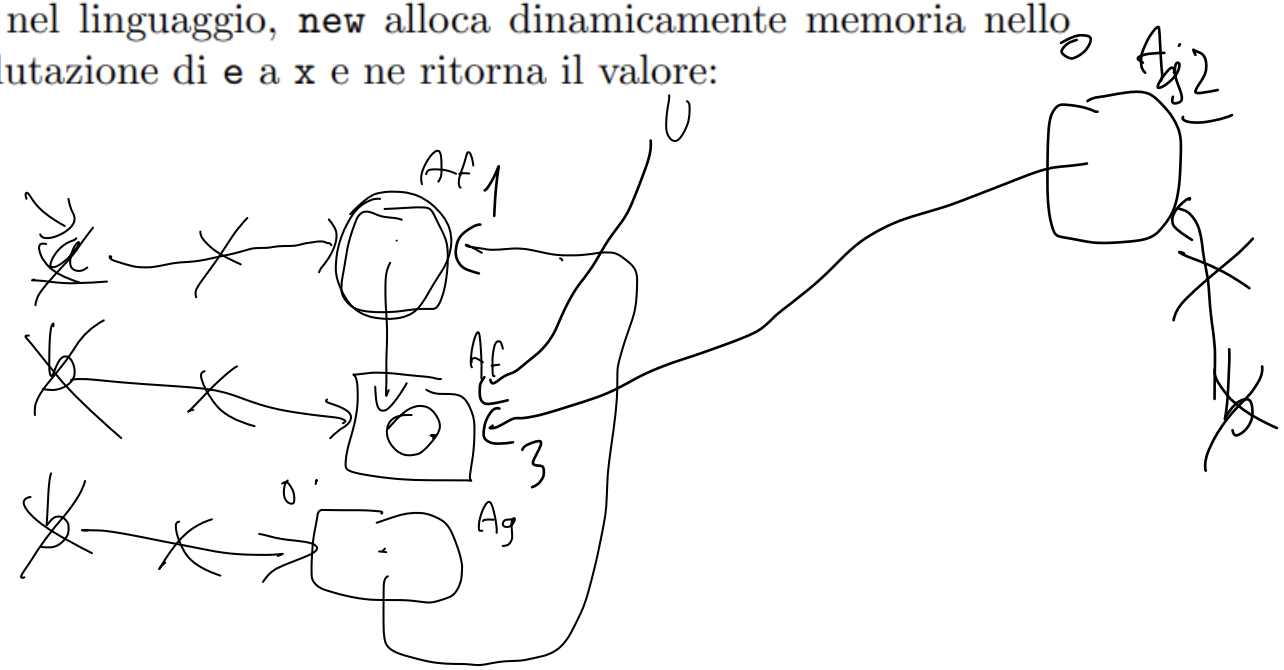
7. Si dica cosa stampa (tramite il comando di stampa in linea, `write( var )`) il seguente frammento di programma in un linguaggio con gestione delle eccezioni e scope statico.

```
int x = 0;
int y = 1;
int z = 0;

void f( int z ) throws W, Z {
 if( x < y ){ throw Z; }
 if ( x > z ){ throw W; }
 write( y );
 y = x + y;
 f( z++ );
}

void g( int z ) {
 try {
  f( z );
 } catch ( Z ){
  write( x );
  x = x + y;
  g( z );
 } catch ( W ){
  write( x );
 }
}

{
 int x = 3;
 g( x );
}
```

Esercizio 4

7. È dato il seguente frammento di codice in un linguaggio con variabili a riferimento e garbage collection con contatori dei riferimenti; nel linguaggio, `new` alloca dinamicamente memoria nello heap e l'espressione `x = e` assegna la valutazione di `e` a `x` e ne ritorna il valore:

```
type A = struct { A next; }
A f(){
 A a = new A();
 A b = a.next = new A();
 return a;
}
A g( A a ){
 A b = new A();
 return b.next = a;
}
A u = g( f() );
u = g( u.next );
```

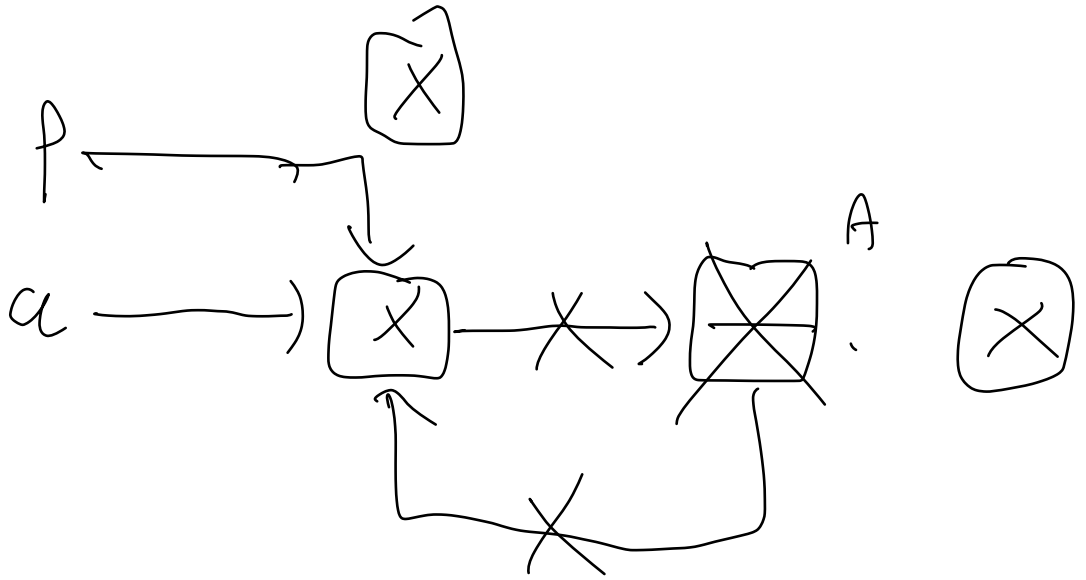


(i) Quanti oggetti di tipo A sono creati sullo heap in totale? (ii) Per ciascuno di essi si dia il valore del contatore dei riferimenti al termine del frammento.

Esercizio 5

8. Il pseudolinguaggio usato nel codice sottostante ammette l'uso dei puntatori—`new A()` alloca una nuova struttura di tipo `A` nello heap e `free( a )` libera la memoria nello heap puntata da `a`. Il codice presenta problemi nella gestione dei riferimenti. Dove? In che modo la tecnica delle "tombstones" risolvere il problema? Motivare la risposta.

```
struct A { struct A* a; int b; };
A* p;
A* a = new A();
*a.a = a;
p = a;
free( p );
*a.b = 5;
```



Esercizio 6

7. Si consideri un linguaggio con passaggio per valore nel quale le eccezioni sono dichiarate con la sintassi `exception E` (`E` nome dell'eccezione), sono sollevate con l'istruzione `throw E` e sono gestite coi blocchi `try { ... } catch E { ... }`. Il linguaggio ha scoping statico per tutti i nomi, eccezioni comprese. Cosa stampa (tramite l'operazione `print`) il seguente frammento? Spiegare brevemente il ragionamento dietro la risposta.

```
exception Y;
a() { throw Y; }
exception X;
b( int x ) {
 if ( x < 5 ){ print( x ); a(); }
 else { print( x ); throw X; }
}
c( int x ) {
 try { b( x++ ); }
 catch ( Y y ) { print( 1 ); c( x ); }
}
d( int z ) { try { c( z ); } catch ( X x ){} }

d( 3 );
```

Esercizio 7

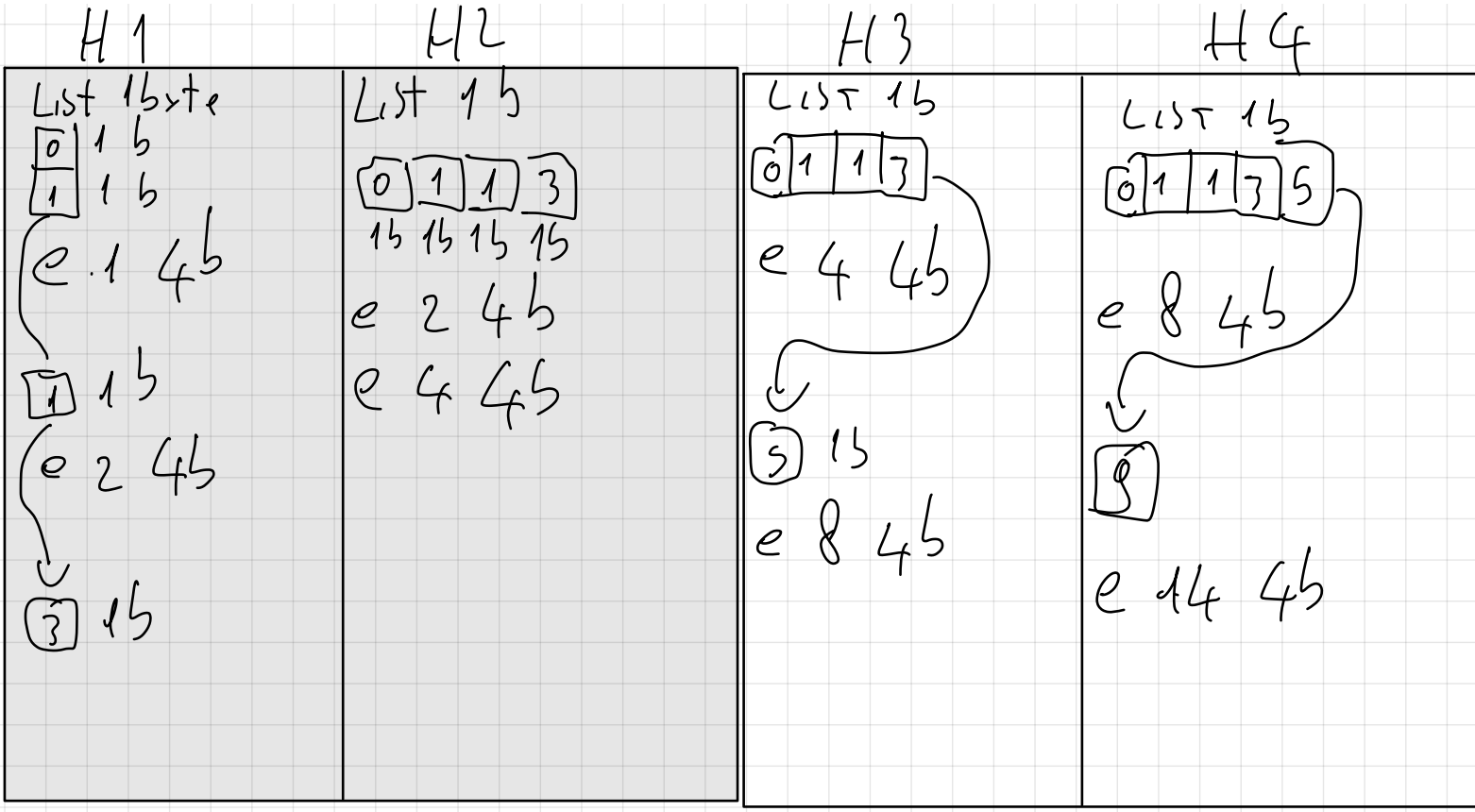
7. In un pseudolinguaggio, `new` crea un nuovo oggetto nello heap. La lista di interi `ListInt` occupa 1 byte, da sommare allo spazio degli `Int` contenuti. Un `Int` occupa 1 byte, mentre un `Long` occupa 4 byte. `ListInt` offre `add`, che aggiunge un intero in coda, e `get`, che fornisce l'intero contenuto alla posizione passata come parametro. Il pseudolinguaggio usa passaggio per riferimento e garbage collection di tipo stop-and-copy, attivando la collection una volta superato l'80% di utilizzo della memoria disponibile al collector. Lo heap ha 30 byte di memoria complessiva ed è assunto inizialmente vuoto. Indicare, spiegando brevemente il ragionamento seguito, quante volte viene chiamato il garbage collector nell'esecuzione del seguente frammento di codice.

```
ListInt acc = new ListInt();
acc.add( new Int( 0 ) );
acc.add( new Int( 1 ) );
for( int i = 2; i < 7; i++ ){
 Long e = new Long( acc.get( i-2 ) + acc.get( i-1 ) );
 if( e % 2 != 0 ){
  acc.add( new Int( e ) );
 } else {
  acc.add( new Int( e + 1 ) );
 }
}
```

7. In un pseudolinguaggio, **new** crea un nuovo oggetto nello heap. La lista di interi **ListInt** occupa 1 byte, da sommare allo spazio degli **Int** contenuti. Un **Int** occupa 1 byte, mentre un **Long** occupa 4 byte. **ListInt** offre **add**, che aggiunge un intero in coda, e **get**, che fornisce l'intero contenuto alla posizione passata come parametro. Il pseudolinguaggio usa passaggio per riferimento e garbage collection di tipo stop-and-copy, attivando la collection una volta superato l'80% di utilizzo della memoria disponibile al collector. Lo heap ha 30 byte di memoria complessiva ed è assunto inizialmente vuoto. Indicare, spiegando brevemente il ragionamento seguito, quante volte viene chiamato il garbage collector nell'esecuzione del seguente frammento di codice.

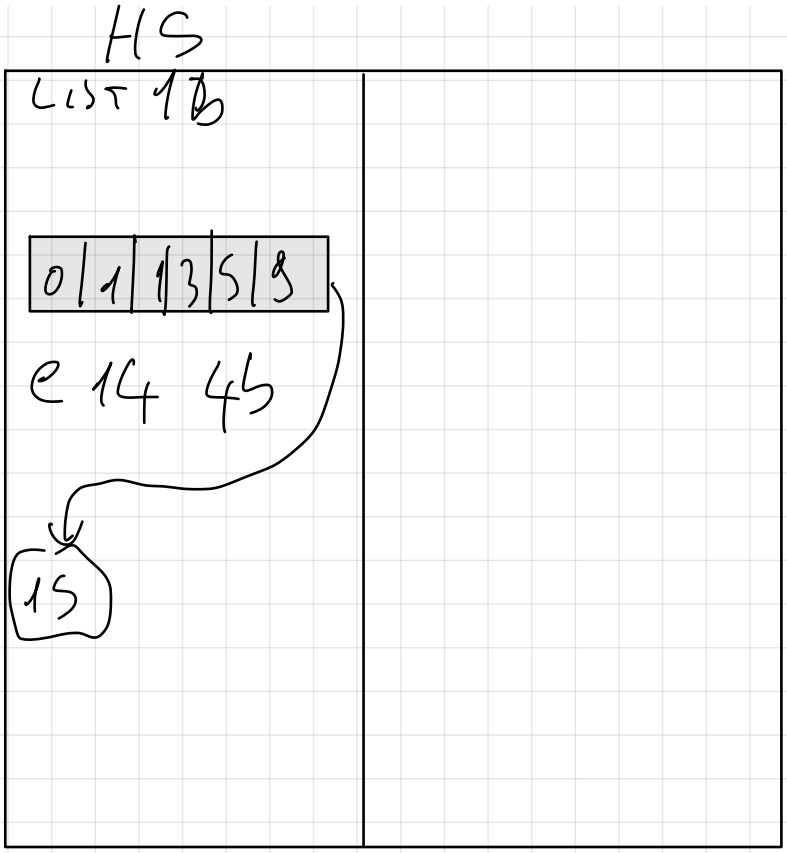
```
ListInt acc = new ListInt();
acc.add( new Int( 0 ) );
acc.add( new Int( 1 ) );
for( int i = 2; i < 7; i++ ){
 Long e = new Long( acc.get( i-2 ) + acc.get( i-1 ) );
 if( e % 2 != 0 ){
  acc.add( new Int( e ) );
 } else {
  acc.add( new Int( e + 1 ) );
 }
}
```

i = 5



7. In un pseudolinguaggio, `new` crea un nuovo oggetto nello heap. La lista di interi `ListInt` occupa 1 byte, da sommare allo spazio degli `Int` contenuti. Un `Int` occupa 1 byte, mentre un `Long` occupa 4 byte. `ListInt` offre `add`, che aggiunge un intero in coda, e `get`, che fornisce l'intero contenuto alla posizione passata come parametro. Il pseudolinguaggio usa passaggio per riferimento e garbage collection di tipo stop-and-copy, attivando la collection una volta superato l'80% di utilizzo della memoria disponibile al collector. Lo heap ha 30 byte di memoria complessiva ed è assunto inizialmente vuoto. Indicare, spiegando brevemente il ragionamento seguito, quante volte viene chiamato il garbage collector nell'esecuzione del seguente frammento di codice.

```
ListInt acc = new ListInt();
acc.add( new Int( 0 ) );
acc.add( new Int( 1 ) );
for( int i = 2; i < 7; i++ ){
 Long e = new Long( acc.get( i-2 ) + acc.get( i-1 ) );
 if( e % 2 != 0 ){
  acc.add( new Int( e ) );
 } else {
  acc.add( new Int( e + 1 ) );
 }
}
```



Esercizio 8

```
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() )
```

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.