PROGRAMMAZIONE II (A,B) - a.a. 2018-19
Prova scritta — 11 Settembre 2019

Domande di Base

1. Si consideri il frammento di programma seguente in un linguaggio imperativo

```
{
    int x=0;
    void p() {
        x=1;
    }
    {
        int x;
        p();
        write(x);
    }
    write(x);
}
```

   Dire cosa stampa il programma in caso venga utilizzata una strategia di scoping dinamico per la soluzione dei riferimenti non locali.

2. Si consideri la struttura del seguente programma a oggetti.

```
public class Bird{
    public void fly(){}
}
public class Anastra extends Bird{}

public class Struzzo extends Bird{}
```

   Dire se il programma viola o meno il principio di sostituzione. Motivare la risposta.

Esercizio 1. Si consideri il nucleo di un semplice linguaggio di programmazione funzionale, la cui sintassi è descritta di seguito:

Posizione $p ::= 0 \mid 1 \mid \ldots \mid 100$

Identificatori $I ::= \ldots$

Espressioni $e ::= I \mid p \mid moveLeft\, e \mid moveRight\, e \mid let\, I = e_1\, in\, e_2$

Intuitivamente, una *posizione* è un tipo di dato che individua una posizione su una retta finita (da 0 a 100). L' espressione *moveLeft* decrementa di uno il valore della posizione passata come argomento. Similmente *moveRight* incrementa di uno il valore della posizione passata come argomento. Il decremento della posizione 0 produce 0 mentre l'incremento della posizione 100 produce come risultato 100.

1. Si definisca l'interprete del linguaggio utilizzando OCaml come linguaggio di implementazione.

Esercizio 2. Si consideri il seguente programma OCaml

```
let x = v;;
let f1 = fun z -> z*x;;
llet rec f2 = fun z-> if z = 1 then f1 x else x * f2 (z-1);;
f2 2;;
```

1. Si dica per quale valore di v il programma precedente produce come risultato il valore 1000.
2. Si descriva lo stato dello stack dei record di attivazione nella simulazione della valutazione del programma.

Esercizio 3. Si consideri il tipo di dato astratto modificabile `Storage`. Assumiamo che `Storage` sia utilizzato per memorizzare e reperire oggetti appartenenti alla classe `DataObject` definita nel modo seguente:

```
class DataObject {
      private String nome;
      private String info;
//opportuni metodi pubblici per accedere e modificare i campi nome e info
}
```

Ogni oggetto inserito nello `Storage` ha associata una *versione*, rappresentata da un intero. Quando un `DataObject` viene inserito nello `Storage`, se non ci sono oggetti con lo stesso nome gli viene assegnata la versione 0; altrimenti gli viene assegnata la massima versione esistente per oggetti con quel nome, più uno. Supponiamo che `Storage` fornisca le seguenti operazioni:

- `put(DataObject dataObj)`: inserisce l'oggetto nello `Storage`, associandogli la versione come descritto sopra. Restituisce la versione associata.
- `get(String s)`: restituisce il `DataObject` di versione piu' recente tra quelli aventi come nome la stringa `s` nello `Storage`, se esiste nello `Storage`.
- `get(String s, int version)`: restituisce il `DataObject` di nome `s` e della versione richiesta, se esiste nello `Storage`.
- `check(String s)`: restituisce `true` se e solo se esiste almeno una versione di un `DataObject` di nome `s` nello `Storage`.

1. Completare la specifica del tipo di dato astratto `Storage` (*overview* con descrizione di un'istanza tipica e specifica completa dei *metodi*, compreso il tipo del risultato ed eventuali eccezioni lanciate).
2. Implementare il costruttore, il metodo `put`, ed i metodi `get` del tipo di dato astratto `Storage`. Utilizzare come struttura di implementazione due *vector*:
   (a) `Vector <DataObject> dataCollection`, contenente oggetti della classe DataObject,
   (b) `Vector <Integer> versioni`, contenente interi che rappresentano la versione del corrispondente `DataObject`.
3. Definire l'invariante di rappresentazione e dimostrare che l'implementazione del metodo `put` preserva tale invariante.