Esercizio 1. Si consideri il seguente programma OCaml:

```
let n = 5;;
let h = fun x -> n + x;;
let rec f p n =
   let g = fun y -> n * y in
      if n = 0 then p 1
      else if n > 1 then f g (n-1)
      else f p (n-1);;
f h 2;;
```

1. Indicare il tipo inferito dall'interprete OCaml per la funzione ricorsiva `f`.
2. Simulare la valutazione del programma mostrando la struttura della pila dei record di attivazione.
3. Indicare il valore restituito dal programma.

Esercizio 2. Si consideri il tipo di dati 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 assengnata 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`..
- `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 dati astratto `Storage` (overview con descrizione di un'istanza tipica e specifica completa dei metodi, compreso il tipo del risultato ed eventuali eccezioni lanciate).
2. Definire una implementazione del tipo di dati astratto `Storage` utilizzando 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.