# PROGRAMMAZIONE II (A,B) - a.a. 2014-15
# Prova scritta — 07 Luglio 2015

**Esercizio 1.** Si consideri il seguente programma scritto in una sintassi C-like.

```
int x = 4;

void printx(void) { printf("%d\n", x); }

void foo(int y) {
  int x = 4;
  int z = 6;
  x = x + z * y;
  printx();
  {
    int x = 0;
    int z = 1;
    x = x + z * y;
    printf("%d\n", x);
    printx();
  }
}

void main(void) {
  int z = 3;
  printx();
  foo(z);
}
```

1. Si descriva lo stato dello stack dei record di attivazione durante le due chiamate della funzione `printx`, nell'ipotesi che l'indirizzo di base del record di attivazione del `main` sia rappresentato dal valore `base_main` e che la dimensione del record di attivazione della funzione `foo` sia rappresentata dal valore `D`.

2. La maggior parte dei compilatori del linguaggio `C` riescono a determinare staticamente la dimensione del record di attivazione di una funzione. Si discuta come sia possibile determinare staticamente una struttura *ottimale* per il record di attivazione della funzione `foo`.

**Esercizio 2.** Si consideri il linguaggio didattico imperativo, e se ne estenda la sintassi astratta e l'interprete del linguaggio in modo gestire un comando iterativo di tipo `for-loop`.

La sintassi concreta del ciclo for è la seguente

```
for (inizializzazione; condizione; incremento/decremento)
    statement
```

dove `inizializzazione` rappresenta una qualsiasi espressione (nella forma più semplice, l'assegnamento di un valore iniziale a un contatore); `condizione` è una condizione che dev'essere verificata affinché il ciclo continui (nella forma più semplice, la condizione che il contatore non abbia raggiunto il suo valore finale); `incremento` (o `decremento`) è una istruzione che viene eseguita al termine di ogni iterazione (nella forma piú semplice, l'incremento del contatore).

**Esercizio 3.** Si consideri il seguente frammento, in una sintassi Java-like, della definizione di una astrazione che intende rappresentare funzioni tra valori interi come opportune collezioni di dati.

```
public class TotalMap {

   public TotalMap() { ... }

   //@effects result!=null
   public Integer get(Integer key) { ... }

   //@requires key!=null && val!=null
   //@effects modify this
   //@effects get(key)==val
   //@effects forall z : z!=key => get(z)==old(get(z))
   public void put(Integer key, Integer val) { ... }

   //@effects result==true
   public boolean containsKey(Integer key) { ... }
}

public class PartialMap {

   public PartialMap() { ... }

   //@effects containsKey(key) <==> result!=null
   public Integer get(Integer key) { ... }

   //@requires key!=null && val!=null
   //@effects modify this
   //@effects get(key)==val
   //@effects forall z : z!=key => get(z)==old(get(z))
   //@effects forall z : has(z) <==> old(has(z)) || z==key
   public void put(Integer key, Integer val) { ... }

   public boolean containsKey(Integer key) { ... }
}
```

1. Senza fare assunzioni sulla struttura dell'implementazione sottostante, ma basandosi solo sulla specifica dei metodi (ove con `result` si indica il valore restituito dal metodo, e con `old` la funzione che restituisce il valore di una espressione prima dell'esecuzione del metodo), si dica se `TotalMap` è un sottotipo di `PartialMap`, ovvero se viene preservato il principio di sostituzione.

2. Supponendo di utilizzare come struttura di implementazione

   ```
   private Vector<Integer> domain;
   private Vector<Integer> codomain;
   ```

   si definiscano per le due classi l'invariante di rappresentazione e l'implementazione dei costruttori.

3. Si fornisca l'implementazione del metodo `put` per la classe `PartialMap` e si dimostri che preserva l'invariante di rappresentazione.