PROGRAMMAZIONE II (A,B) - a.a. 2013-14
Seconda Valutazione Intermedia — 30 maggio 2014

Esercizio 1. Si consideri il seguente programma a oggetti scritto in una sintassi Java-like:

```
class B {
     public void foo(B bar) {
          System.out.print("B1 ");
     }
     public void foo(C car) {
          System.out.print("B2 ");
     }
}
class C extends B {
     public void foo(B barba) {
          System.out.print("C1 ");
     }
     public void foo(C carca) {
          System.out.print("C2 ");
     }
}
```

```
class Main {
     public static void main(String[] args) {
          (1) B c = new C();
          (2) B b = new B();
          (3) b.foo(c);
          (4) c.foo(b);
          (5) c.foo(c);
      }
}
```

1. Cosa stampa il programma?
2. Descrivere le informazioni presenti a run-time al termine dell'esecuzione del comando (1), e cioè il contenuto della class area, delle tabelle dei metodi, del run-time stack e dello heap.
3. Mostrare la struttura dei record di attivazione che vengono creati sullo stack in corrispondenza dell'invocazione del metodo `foo` nelle istruzioni (3), (4) e (5).

Esercizio 2. Per questo esercizio, si scelga uno solo dei due programmi seguenti, scritti rispettivamente in JavaScript e in OCaml ma sostanzialmente equivalenti. Indicare in modo chiaro la scelta fatta.

```
function dummy(x) {
    return 1000;
}
function evil(g,n) {
    function f(x) {
        return 10 + n;
    }
    if (n == 1) return f(0) + g(0); (*)
    else return evil(f,n-1);
}
evil(dummy,2);
```

```
let dummy x = 1000 in
    let rec evil g n =
                (let f x = 10 + n in
                 if n = 1 then f 0 + g 0  (*)
                          else evil f (n - 1))
    in evil dummy 2
```

1. Determinare il tipo di tutti gli identificatori che compaiono nel programma scelto.
2. Descrivere lo stato dello stack dei record di attivazione subito dopo l'invocazione della funzione g nella linea (*). Indicare per ogni record di attivazione le informazioni relative al puntatore di catena statica, puntatore di catena dinamica e struttura dell'ambiente locale.
3. Qual è il valore calcolato dal programma?

Esercizio 3. Si consideri il linguaggio didattico imperativo senza funzioni né procedure. Estendiamo la sintassi dei comandi in modo da includere il comando condizionale multiplo **case-esac**, con la seguente sintassi concreta:

```
case
| E1 -->  CL1
| E2 -->  CL2
...
| En -->  CLn
esac
```

Nel comando, per ogni $i \in \{1, 2, \ldots, n\}$, `Ei` è una espressione booleana chiamata *guardia*, mentre `CLi` è una lista di comandi. Il numero n è un intero maggiore o uguale a zero.

L'esecuzione del comando case-esac consiste nel valutare sequenzialmente le guardie, e per ogni guardia che restituisce true nell'eseguire la corrispondente lista di comandi.

1. Estendere la sintassi astratta del linguaggio didattico imperativo in modo da includere il comando condizionale multiplo **case-esac**.

2. Definire le regole di semantica operazionale della valutazione del comando case-esac e derivare il relativo codice OCaml per interpretare il comando. Si usino liberamente le funzioni OCaml:

```
semc: com * (dval env) * (mval store) -> (mval store)
semcl: (com list) * (dval env) * (mval store) -> (mval store)
sem:  exp * (dval env) * (mval store) -> eval
```

Esercizio 4. La classe Java Stack nel box a sinistra implementa una semplice pila con i metodi pop() e push(), utilizzando un *array parzialmente riempito*. Uno studente osserva che il codice della pop() può essere semplificato, e realizza la sottoclasse CleverStack mostrata nel box a destra.

```
public class Stack{
    Object [] stack = new Object[1000];
    int size = 0;

    void push(Object obj){
        stack[size] = obj;
        size++;
    }
    Object pop(){
        if (size == 0) return null;
        else {
            size--;
            Object tmp = stack[size];
            stack[size] = null;
            return tmp;
        }
    }
}
```

```
class CleverStack extends Stack{
    Object pop(){
        if (size == 0) return null;
        else {size--;
              return stack[size]};
    }
}
```

1. Confrontare in modo informale il comportamento delle due classi rispetto all'occupazione di memoria a run-time.

2. Fornire un programma che usa un oggetto di tipo Stack, e tale che se lo si sostituisce con un oggetto di tipo CleverStack si ottiene un comportamento diverso.