Linguaggi e Programmazione Orientata agli Oggetti

a.a. 2020/2021

20 gennaio 2022

1. (a) Per ogni stringa elencata sotto stabilire, motivando la risposta, se appartiene alla seguente espressione regolare e, in caso affermativo, indicare il gruppo di appartenenza (escludendo il gruppo 0).

   Nota bene: la notazione `(?: )` permette di usare le parentesi in un'espressione regolare **senza** definire un nuovo gruppo.

   ```
   ([a-z]+(?:\.[a-z]*)*)|(0[bB][0-1]+[lL]?)|(\s+)|(mv\.ACC|add\.SP|sub\.FM)
   ```

   i. `"mv.ACC"`
   ii. `"MV.acc"`
   iii. `"0B"`
   iv. `"b."`
   v. `"0BL"`
   vi. `"0B10L"`

   (b) Mostrare che la seguente grammatica è ambigua.

   ```
   Exp ::= Term | Exp + Term | - Exp
   Term ::= a | b | - Term | ( Exp )
   ```

   (c) Modificare la grammatica definita al punto precedente in modo che **non sia ambigua** e che il linguaggio generato a partire dal non terminale `Exp` **resti invariato**.

2. Sia `sum_wise : int list -> int list -> int list` la funzione tale che

   `sum_wise` $[a_0;\ldots;a_n]$ $[b_0;\ldots;b_n]$ = $[a_0 + b_0;\ldots;a_n + b_n]$

   e `sum_wise` l_1 l_2 solleva un'eccezione (usare `raise (Invalid_argument "sum_wise")`) se l_1 e l_2 non hanno la stessa lunghezza.

   Esempi:

   ```
   sum_wise [0;2;4] [1;3;5]=[1;5;9]
   sum_wise [] []=[]
   sum_wise [0] [1;3;5] solleva un'eccezione
   sum_wise [0;2;4] [1] solleva un'eccezione
   ```

   (a) Implementare `sum_wise` senza uso di parametri di accumulazione.

   (b) Implementare `sum_wise` usando un parametro di accumulazione affinché la ricorsione sia di coda.

3. Completare il seguente codice che implementa

   - gli alberi della sintassi astratta `AST` composti da nodi corrispondenti ai literal di tipo booleano (`BoolLit`) e all'operatore di congiunzione (`And`);
   - il visitor `Eval` che valuta l'espressione nel modo convenzionale restituendo un booleano come risultato.

   Esempio:

   ```
   var exp = new And(new BoolLit(true), new BoolLit(true));
   var v = new Eval();
   assert exp.accept(v); // true && true si valuta in true
   exp = new And(new BoolLit(true), new BoolLit(false));
   assert !exp.accept(v); // true && false si valuta in false
   ```

Codice da completare:

```
import static java.util.Objects.requireNonNull;
public interface AST { <T> T accept(Visitor<T> v); }

public interface Visitor<T> {
  T visitBoolLit(boolean b);
  T visitAnd(AST left,AST right);
}

public class BoolLit implements AST {
  private final boolean b;
  public BoolLit(boolean b) {
    // completare
  }
  @Override
  public <T> T accept(Visitor<T> v) {
    // completare
  }
}

public class And implements AST {
  // invariant left != null && right !=null
  private final AST left;
  private final AST right;
  public And(AST left,AST right) {
    // completare
  }
  @Override
  public <T> T accept(Visitor<T> v) {
    // completare
  }
}

public class Eval implements Visitor<Boolean> {
  @Override
  public Boolean visitBoolLit(boolean b) {
    // completare
  }
  @Override
  public Boolean visitAnd(AST left, AST right) {
    // completare
  }
}
```

4. Considerare le seguenti dichiarazioni di classi Java:

```
public class P {
  String m(float f1,float f2) { return "P.m(float,float)"; }
  String m(float[] f) { return "P.m(float[])"; }
}
public class H extends P {
  String m(double d1,double d2) { return "H.m(double,double)"; }
  String m(double[] d) { return "H.m(double[])"; }
}
public class Test {
    public static void main(String[] args) {
        P p = new P();
        H h = new H();
        P p2 = h;
        System.out.println(...);
    }
}
```

Dire, per ognuno dei casi elencati sotto, che cosa succede sostituendo al posto dei puntini nella classe `Test` il codice indicato, assumendo che tutte le classi siano dichiarate nello stesso package.

Per ogni caso fornire due o tre righe di spiegazione così strutturate: se c'è un errore in fase di compilazione, specificare esattamente quale; se invece la compilazione va a buon fine spiegare brevemente perché e descrivere cosa avviene al momento dell'esecuzione, anche qui spiegando brevemente perché.

(a) `p.m(42.0f,42.0f)`

(b) `p2.m(42.0f,42.0f)`

(c) `h.m(42.0f,42.0f)`

(d) `p.m(42,42)`

(e) `p2.m(42,42)`

(f) `h.m(42,42)`