PROGRAMMAZIONE II (A,B) - a.a. 2018-19
Quarto Appello – 15 Luglio 2019

Domande di base

1. Cosa si intende per *dynamic dispatch* nel contesto dei linguaggi di programmazione a oggetti?
2. Si consideri il seguente programma (in un opportuno linguaggio di programmazione)

```
function myf(int a, int b, int c) {
    a := b + c;
    b := c + 1;
    print a, b, c;
  }

function main {
    int i := 5;
    int j := 10;
    ink k := 15;
    myf(i, j, j + k);
    print i, j, k;
  }
```

   Quali valori vengono stampati dall'esecuzione del programma nel caso in cui i parametri siano trasmessi per nome?

3. Si consideri un linguaggio di programmazione con funzioni che adotta la regola di scoping statico. Cosa si intende per *ambiente non locale di una funzione*?

Esercizio 1

Si consideri un tipo di dato astratto *PriorityQueue* che rappresenta una collezione di elementi ordinati in modo decrescente in base ad una priorità associata agli elementi. La priorità di ogni elemento varia tra 0 e n-1 (n parametro). La politica standard (LIFO) della coda vale solo rispetto alle occorrenze di elementi con la stessa priorità. L'interfaccia `PriorityQueue<E>` è generica rispetto al tipo `E` degli elementi e deve includere, tra gli altri, i seguenti metodi

- `public void add(E el, int p)` *il cui effetto è quello di inserire* `el` *nella coda con priorita p*
- `public E peek( )` *il cui effetto è quello di restituire l'elemento con priorità minima contenuto nella coda*
- `public E poll ()` *il cui effetto è quello di restituire l'elemento con priorità minima contenuto nella coda e di rimuoverlo*
- `public int size ()` *restituisce il numero di elementi contenuti nella coda*
- `public int min ()` *restituisce la priorità minima tra gli elementi contenuti nella coda*

1. Si definsca la specifica dell'interfaccia `PriorityQueue<E>`, indicando per ogni metodo le clausole REQUIRES, MODIFY ed EFFECTS, il valore restituito e le eventuali eccezioni lanciate in dipendenza dei parametri attuali.

2. Si assuma di implementare la classe `MyPriorityQueue<E>` con le seguenti variabili d'istanza:

```
private int max;
private ArrayList<Pair<E>> coda;
```

dove `Pair<E>` definisce coppie generiche elemento-priorità .

```
public Pair<E> {
   public E el;
   public int p;
   public Pair<E>(E el, int p )
   {this.el=el; this.p=p}
   }
```

- Si definiscano la funzione di astrazione e l'invariante di rappresentazione.
- Si implementi il costruttore ed il metodo `add` verificando che preservino l'invariante di rappresentazione.

Esercizio 2

Si consideri il linguaggio didattico funzionale, e se ne estenda la sintassi astratta e l'interprete in modo da gestire un costrutto `sum-all` che applica una funzione non ricorsiva a tutti i valori presenti in un albero binario di valori interi.

1. La sintassi concreta del costrutto è la seguente `sum-all(albero,funzione)`. Il costrutto restituisce come risultato la somma dei valori calcolati dall'applicazione della funzione;
2. Una albero binario puòessere in modo modo standard ovvero come una epressione `ETree of tree` dove `tree = Empty | Node of tree * exp * tree`.

Esercizio 3

Si consideri il seguente programma OCaML

```
let x =1;;

let f1  = fun y z  ->  let  f2 = fun x  ->    x * (y+ z)
in f2 x * (y-z);;

let reverse lst = let rec aux app tmp =
                        match tmp with
                            | [] -> app
                            | h::t -> aux (h::app) t
                  in aux [] lst

let rec apply g  n  (lst : int list) =
                                  match (reverse lst) with
                                   |[] -> []
                                   | hd::ls ->  (g hd n):: apply g n ls;;

let res  = apply  f1  (x+1 )  [2;5;1];;
```

1. Si simuli la valutazione del programma mostrando la struttura della pila dei record di attivazione.
2. Si determini il valore calcolato dal programma.