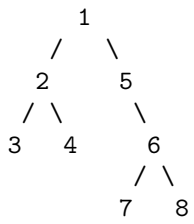
Programmazione ad Oggetti
Mod. 2

13/9/2022

Studente _____ Matricola _____

1. Vogliamo realizzare in Java 8+ una classe `TreeNode`, parametrica su un tipo generico `T`, che rappresenta nodi di un albero binario decorati con valori di tipo `T`. Quando entrambi i sotto-alberi sinistro e destro di un nodo sono assenti, allora il nodo rappresenta una foglia. Seguono ora le specifiche dettagliate.

   (a) [7 punti] Gli alberi devono essere *iterabili* e l'iteratore deve attraversare l'albero in DFS (*Depth-First Search*), fornendo gli elementi di tipo `T` in *pre-ordine*, ovvero nell'ordine mostrato da questo esempio:

   (b) [4 punti] Gli alberi devono essere confrontabili tramite il metodo `equals()`. Due alberi sono uguali se tutti i sotto-alberi sono uguali e sono costituiti da elementi uguali.

   (c) [4 punti] Si definiscano gli opportuni costruttori ed eventualmente dei metodi statici che fungono da pseudo-costruttori. L'obiettivo è fare in modo che gli alberi siano facili da costruire innestando ricorsivamente le chiamate. Non si permetta l'istanziazione di alberi vuoti o non inizializzati da popolare successivamente con dei setter.

   (d) [2 punti] Si implementino uno o più snippet di test che mettono alla prova tutte le caratteristiche richieste.

   (e) [2 punti (bonus)] Si implementi un *pretty printer* tramite un override del metodo `toString()`.

   Si implementi la classe `TreeNode` secondo le specifiche date, rappresentando la struttura dati nella maniera che si ritiene più conveniente e fornendo tutti i metodi necessari. È importante il riuso di codice, l'information hiding ed una implementazione che escluda il più possibile stati di invalidità grazie ad un saggio uso dei tipi.

2. Vogliamo realizzare in C++ lo stesso sistema di alberi binari di cui al Quesito 1. La traduzione del codice da Java a C++ non deve essere letterale: è necessario adottare gli stili e le convenzioni di C++ e di STL, ad esempio formulando il confronto tramite l'overloading degli opportuni operatori, definendo gli opportuni *member type* per gli iteratori e implementando gli overload const e non-const dei metodi laddove necessario. La classe `tree_node` deve avere un template parameter `T` ed aderire allo stile della *value*-oriented programming, comportandosi come un valore con gli opportuni costruttori e operatori di assegnamento. Segue la specifica dettagliata.

   (a) [6 punti] Gli alberi devono essere *iterabili* e l'iteratore deve attraversare l'albero in maniera algoritmicamente equivalente all'implementazione Java di cui al Quesito 1.(a). Si definiscano i *member type* per iteratori const e non-const e le relative coppie di metodi `begin()` ed `end()`. Se necessario, implementare gli iteratori tramite classi ausiliarie.

   (b) [3 punti] La definizione di equivalenza di due alberi è semanticamente equivalente a quella definita nel Quesito 1.(b). Si assuma che gli oggetti di tipo `T` siano confrontabili tramite l'operatore ==.

   (c) [3 punti] Si definiscano gli opportuni costruttori, operatori di assegnamento ed eventuali distruttori secondo lo stile della *value*-oriented programming. Se vantaggioso, si forniscano metodi statici che fungano da pseudo-costruttori e permettano di costruire alberi facilmente innestando ricorsivamente le chiamate.

(d) [2 punti] Si implementino uno o più snippet di test.

(e) [2 punti (bonus)] Si implementi un *pretty printer* tramite un overload globale dell'operatore <<.

Si utilizzi la revisione del linguaggio C++ che si preferisce, purché si specifichi quale. È importante fare buon uso del type system, riusando il codice laddove possibile.

Total for Question 2: 14

Question:	1	2	Total
Points:	17	14	31
Bonus Points:	2	2	4
Score:			