Daniele Dituri Matr: 873401 d.dituri@campus.unimib.it

Relazione progetto Programmazione C++

Progetto C++

Introduzione

Il progetto consiste nella realizzazione di una classe generica che implementa un albero binario di ricerca. L'albero è formato da un insieme di elementi T contenuti in nodi connessi in una struttura gerarchica padre-figlio e NON deve permettere l'inserimento di dati duplicati. Deve essere possibile per l'utente scegliere liberamente la strategia usata per confrontare due dati T.

Oltre ai metodi fondamentali, la classe deve permettere:

1. Costruire un albero anche con un costruttore secondario che prende in input due iteratori generici di inizio e fine sequenza di dati
2. Conoscere il numero totale di dati inseriti nell'albero
3. Controllo di esistenza un elemento T
4. Accedere ai dati presenti nell'albero tramite un iteratore a sola lettura e di tipo forward
5. Stampare il contenuto dell'albero usando operator<<
6. Implementare inoltre un metodo subtree che, passato un dato d dello stesso tipo del dato contenuto nell'albero, ritorna un nuovo albero. Il nuovo albero deve corrispondere al sottoalbero avente come radice il nodo con il valore d
7. Implementare una funzione globale printIF che dato un albero binario di tipo T, e un predicato P, stampa a schermo tutti i valori contenuti nell'albero che soddisfano il predicato.

Scelte progettuali

Il progetto contiene due file main.cpp e bst.hpp

main.cpp

Rappresenta il file di test. In questo file sono presenti diversi metodi per testare tutte le funzioni di bst.hpp con tipi di dati differenti. Per testare la correttezza completa della classe generica sono stati creati funtori di ordinamento e uguaglianza su int, char e sul tipo composto team (formato dal nome della squadra e dalla posizione in classifica). Per il tipo team i funtori controllano la posizione delle squadre mandate. È stato aggiunto inoltre l'operatore di output sempre per team per permettere al programma di stampare a schermo nel modo corretto.

Tra le altre scelte prese c'è la creazione di altri quattro funtori utili per testare il metodo globale printIF. I quattro funtori sono:

- is_even: controlla che un int sia un numero pari
- grt50: controlla che un int sia un numero maggiore di 50
- is_vowel: controlla che un char sia una vocale
- cmpl: controlla che il team sia arrivato in Champions League, ovvero che sia arrivato tra le prime 4 posizioni

bst.hpp

Rappresenta la classe generica templata che implementa un albero binario di ricerca. Tra le scelte più importanti c'è la gestione della copia, della distruzione e della stampa in modo ricorsivo. È stata scelta

questa strada per facilitare una visita completa dell'albero in tutti i suoi rami nel modo più semplice possibile.

Un'altra scelta particolare riguarda l'operatore di pre e post incremento del forward iterator. Per evitare di spostarsi su nodi già visitati nell'albero viene controllato prima che non sia presente un nodo a destra del nodo corrente. Se presente allora l'iteratore si sposterà nel nodo più a sinistra di questo nodo di destra. Al contrario se non è presente alcun nodo di destra l'iteratore risale nell'albero fino a che non esiste un padre del nodo corrente e evitando però i nodi padre che hanno come figlio destro proprio il nodo corrente (evito quindi il padre del nodo corrente, poiché essendo nel nodo appena a destra il nodo padre è già stato visitato in precedenza).

Infine, l'ultima scelta presa è stata quella di impostare il metodo begin dell'iteratore in modo tale che ti ritorni il nodo più a sinistra possibile, ovvero il nodo più piccolo.

Progetto Qt

Introduzione

Il progetto richiede la creazione di un sistema per la risoluzione automatica di un Sudoku. Il Sudoku è costituito da una griglia di numeri 9 x 9, e l'intera griglia è anche divisa in caselle 3 x 3.

Devono essere implementate quindi le seguenti funzionalità:

1. Visualizzare una griglia 9 x 9 vuota
2. Consentire all'utente di modificare il contenuto delle celle per poter inizializzare le cifre nella griglia del Sudoku
3. Avere un pulsante "Risolvi" per avviare l'algoritmo di risoluzione
4. Visualizzare la griglia del Sudoku risolta o un messaggio d'errore nel caso in cui non esiste soluzione (il contenuto della griglia a questo punto non deve essere modificabile)
5. Visualizzare la griglia del Sudoku risolta o un messaggio d'errore nel caso in cui non esiste soluzione (il contenuto della griglia a questo punto non deve essere modificabile)
6. Avere un pulsante per resettare la griglia del Sudoku e procedere con un nuovo inserimento

La risoluzione deve essere fatta tramite un algoritmo di backtracking fatto nel seguente modo:

- Ricorsivamente cerca quali celle della griglia sono vuote
- Quando trova una cella vuota, la riempie con una cifra nel range consentito (punto 1 delle regole) e controlla se è valida o meno (se soddisfa il punto 2 delle regole)
- Se non è valida, verifica la presenza di altri numeri
- Se tutte le cifre nel range consentito sono state verificate e non è stata trovata alcuna cifra valida da posizionare, si torna all'opzione precedente.

Scelte progettuali

Il progetto contiene tre file main.cpp, mainwindow.cpp e myvalidator.cpp, questi ultimi due con, inoltre, il loro file header. Nel file main.cpp non è stato aggiunto nulla oltre alle istruzioni predefinite

mainwindow

mainwindow.cpp è il file contente tutti i layout e widget del sudoku. Il layout è stato diviso in un layout verticale principale con all'interno un layout a griglia per il sudoku (sudokuLayout), i bottoni Resolve e Clear e un ulteriore layout orizzontale per i bottoni (buttonLayout) Next e Previous.

sudokuLayout contiene all'interno altri 9 layout a griglia (singleSudokuLayout) utilizzati per formare i 9 quadrati del sudoku. Ognuno di questi sotto layout si trova all'interno di un QFrame, questo è stato utile per creare un bordo spesso attorno ad ogni sotto quadrato. Ogni singleSudokuLayout contiene poi 9 QLineEdit utilizzati come celle del sudoku.

Le celle del sudoku sono state impostate in modo tale da avere una lunghezza del tasto di massimo un carattere e supportano solamente numeri compresi da 1 a 9 grazie al validatore creato in myvalidator. Le celle sono state dichiarate inoltre nel file .h di mainwindow in modo tale da renderle accessibili a tutti i metodi che lo necessitano.

Tra le scelte particolari c'è la presenza del metodo on_celltextChanged che in caso di scrittura manuale in una cella di un numero errato per le regole del sudoku colorerà questo numero di rosso.

Tra i metodi ci sono tutti i metodi relativi al click di un bottone, un metodo che cancella tutte le celle del sudoku, uno che controlla la correttezza delle celle inserite manualmente prima di risolvere il sudoku, uno che risolve il sudoku e uno che imposta le celle del sudoku in sola lettura dopo aver premuto il tasto risolvi.

myvalidator

Il file myvalidator è stato creato per l'esigenza di rifiutare il numero 0 nelle celle del sudoku. Inizialmente era stato scelto il QIntValidator accettando solo i numeri compresi tra 1 e 9, quest'ultimo accetta però per esempio il numero 07, avendo però bloccato le celle ad un singolo carattere il risultato era quello di avere delle celle con il numero 0.

Al suo interno è presente solamente un metodo validate che ritornata un validator accettabile se il numero è compreso tra 1 e 9, intermedio se è la cella è vuota e invalido altrimenti