GUI для лабораторных по деревьям

Задачи

1. Двоичное дерево

1. Исправить форму (`TreeWidget.ui`):

- добавить недостающую кнопку `Remove` для удаления указанных ключей из дерева;
- скомпоновать виджеты таким образом, чтобы при изменении размера окна дочерние виждеты адаптировались под это изменение (все кнопки/поля должны быть видны при любом размере окна).

2. Связать сигналы кнопок с формы со слотами класса `TreeWidget` в конструкторе (определите пустые слоты, если необходимо, на данном этапе). В качестве примера в конструкторе уже реализовано одно соединение.
3. Реализовать случайную генерацию ключей: При нажатии кнопки `Randomize` поле с ключами (`lineEdit_keys`) должно быть заполнено случайными ключами в диапазоне от `min` до `max` . Число ключей - `max - min + 1` . Прежнее содержимое поля удаляется. Ключи не должны повторяться. Для примера подобной генерации изучите метод `BinaryTree::buildRandom` .
   Примечание: ключи добавляются **только** в текстовое поле! Добавить ключи в дерево можно с помощью нажатия кнопки `Add` .
4. Заменить файлы `BinaryTree.h` и `BinaryTree.cpp` вашей реализацией двоичного дерева.
5. Реализовать логику слотов класса `TreeWidget` , оставшихся нереализованными после п.2.

Ожидаемый результат

1. UI программы адаптируется под размер окна; нет перекрытия/сокрытия элементов интерфейса при изменении размеров окна.
2. При нажатии кнопки `Randomize` поле с ключами (`lineEdit_keys`) должно быть заполнено случайными ключами в диапазоне от `min` до `max` . Число ключей - `max - min + 1` . Прежнее содержимое поля удаляется. Ключи не должны повторяться.
3. При нажатии кнопки `Add` ключи из текстового поля добавляются в дерево.
4. При нажатии кнопки `Remove` ключи из текстового поля удаляются из дерева.
5. При нажатии кнопки `Clear` выполняется удаление всех узлов из дерева.

Примечание: при любом изменении дерева пользователь должен сразу же увидеть это изменение на экране.

2. Двоичное дерево поиска

1. Добавить файлы дерева поиска в проект.
2. Добавить на форму выпадающий список (`Input Widgets - Combo Box`). С помощью этого списка реализуется переключение между деревьями (обычным и поиска).
3. Реализовать слот в классе `TreeWidget` , который будет осуществлять переключение. Если при переключении дерево непусто, заполнить новое дерево ключами из старого (рекомендуется использовать обход по уровням для равномерного распределения ключей - код есть в лекции по двоичному дереву).
   Примечание 1: хранение дерева в классе `TreeWidget` реализовать через указатель на базовый класс.
   Примечание 2: у класса `QComboBox` существует две перегрузки метода `currentIndexChanged` с разными параметрами: `int` и `QString` . Для выбора нужной перегрузки при вызове функции `connect` используйте функцию qOverload или метод QOverload::of.

Ожидаемый результат

1. Не должна быть нарушена работа пунктов из задачи №1.
2. При выборе пункта выпадающего меню `Binary search tree` выполняется перестройка текущего дерева по правилам двоичного дерева поиска.
3. При выборе пункта выпадающего меню `Binary tree` выполняется перестройка текущего дерева по правилам обычного двоичного дерева.