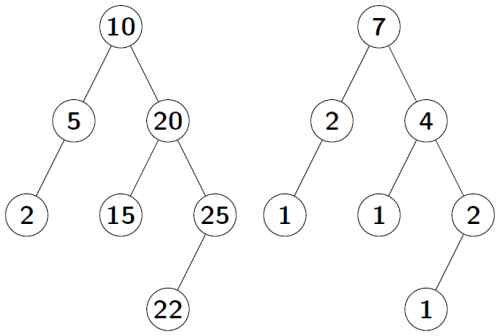
Задачи:

За следващите задачи използвайте създаденият клас **BinaryTree** от миналия път

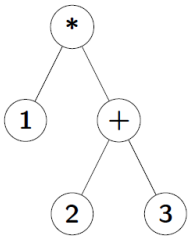
1. Създайте **конструктор** на BinaryTree, който по дадено число **h** да построи идеално балансирано дърво с височина h. Стойноттта на ключовете на елементите на дървото да е равна на нивото, в което се намира елемента.
2. Създайте предикат **isOrdered**, който връща true, ако дървото е наредено и false – ако не е.
3. Създайте предикат **isBalanced**, който вреща true, ако дървото е балансирано и false – ако не е.
4. Създайте метод **mirrorTree**, който обръща дървото огледално.
5. Създайте метод **childrenify**, който заменя стойноттта на ключа на всеки възел V от дървото с числото, съответстващо на броя на всички елементи на поддървото, на което разглеждания връх V е корен. При операцията всеки от възлите да бъде посетен най-много веднъж!



За следващите задачи може да ползвате класическата реализация на BinaryTree, където възлите са просто елементи от тип T

6. Да се реализира метод **parseExpression(std::string s)**, който по правилно построен израз, записан в низа s, създава двоично дърво от символи, представящо израза по следното правило:
    a. Ако изразът е от типа “x”, където x е цифра, то съответното му дърво е листо със стойност символа x.
    b. Ако изразът е от типа “(<израз 1><op><израз 2>)”, то съответното му дърво има като стойност на корена символа на съответния оператор, ляво поддърво - съответно на <израз 1> и дясно поддърво - съответно на <израз 2>.

Пример: s = (1*(2+3))



7. Да се реализира метод **calculateExpressionTree**, който намира стойноттта на израз по дадено дърво, построено от предишния метод.