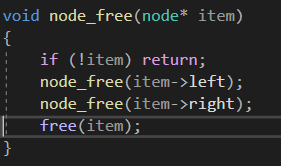
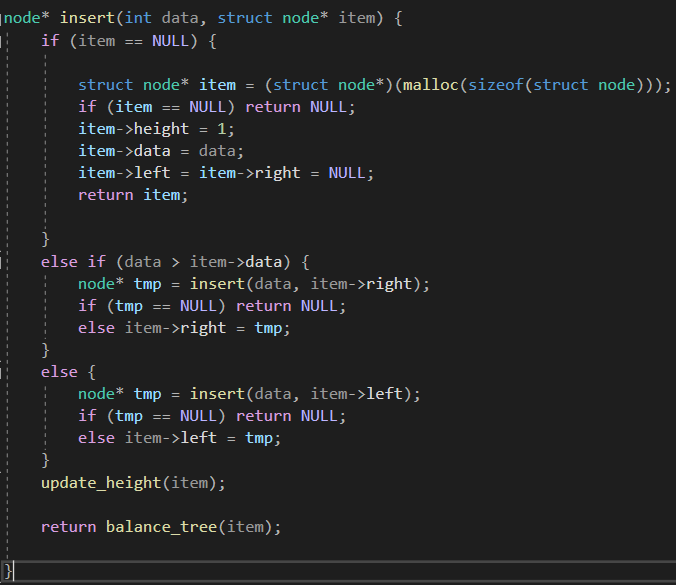
**AVL-TREE** (доступ ко всем элементам за logN)

Остановился на варианте с выделением памяти под все узлы сразу(т.к. по памяти не влезает).

Если рассматривать случай с выделением под каждый узел, то мы бы выделили память под каждый узел, если это невозможно, то выход из insert в main -> “Error memory allocation!” и очистка дерева таким способом:

Ну а в нашем случае можно обойтись одним free для очистки дерева.

**Добавление в AVL-TREE**

Основная особенность AVL дерева в том, что для каждой вершины высота её двух поддеревьев различается не более чем на 1. Поэтому при каждом добавлении стоит учитывать это условие и в случае невыполнения делать балансировку.

Функция insert возвращает корень уже сбалансированного дерева.

Вершины в AVL дереве выстраиваются как и в бинарном дереве поиска( т.е если вершина меньше данного корня, то идем в левое поддерево, если больше, то в правое, продолжаем пока не найдем свободное место) После такой вставки делаем балансировку.

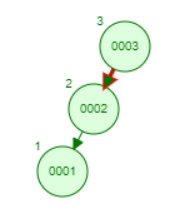
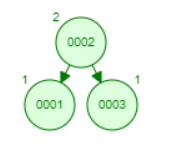
Рассмотрим случаи разбалансировки и их балансировку:

1. Перевес вправо -> нужно сделать малое левое вращение
2. Перевес вправо и у правого потомка перекос влево -> нужно сделать большое левое вращение(состоит из правого вращения для правого потомка, а потом левого для корня)
3. Перевес влево -> нужно сделать правое малое вращение
4. Перевес влево и у левого потомка перекос вправо -> нужно сделать большое правое вращение(состоит из левого вращения для левого потомка, а потом правого для корня)

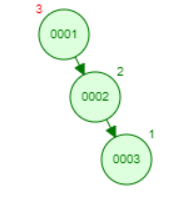
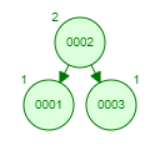
Чтобы распознать разбалансировку используем функцию balance\_factor, которая возвращает разницу между высотой правого и левого поддерева.

Повороты:

* *Левый*

* Правый

Вопросы и ответы:

1) Опишите своими словами:

   а) В чем польза от этих АВЛ деревьев?

   б) Зачем и в каком случае Вы бы их стали использовать в программе?

   в) Чем это лучше чем просто массив или связный список?

1а)  Дерево всегда отсортировано(слева направо). Поиск элемента или проверка на его существование в дереве O(logN).

1б) Если требуется хранить какую-нибудь структуру, в которой часто нужно было бы производить поиск элементов. Это позволит уменьшить время их поиска.

1в)Добавление + сортировка и получение из массива/списка займет больше времени.

Поиск в списке займет O(N) времени, с массивом вставка/удаление O(N + logN) - logN для поиска и N для сдвига( при этом массив должен быть отсортирован) + выделение куска памяти под новый массив.