Санкт-Петербургский политехнический университет Петра Великого

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**Отчёт по лабораторной работе №5**

**Дисциплина**: Низкоуровневое программирование

**Тема:** Программирование на C

Выполнил студент гр. 3530901/90003 В.М. Ковалевский

(подпись)

Преподаватель А.О. Алексюк

(подпись)

“ ” 2021 г.

Санкт-Петербург

2021

**СОДЕРЖАНИЕ**

[Техническое задание 3](#_Toc355724179)

[Описание задачи 4](#_Toc846789292)

[Описание API библиотеки 5](#_Toc1179336432)

[Описание формата ввода 7](#_Toc554317866)

[Сборка библиотеки и программ 9](#_Toc1737982798)

# Техническое задание

Разработать статическую библиотеку, реализующую определенным вариантом задания абстрактный тип данных.

Разработать демонстрационную программу - консольное приложение, обеспечивающее ввод данных с консоли, их обработку и вывод.

Вариант задания: Красно-Черное дерево.

# Описание задачи

Красно-Черное дерево - один из видов самобалансирующихся двоичных деревьев поиска, гарантирующих логарифмический рост высоты дерева от числа узлов и позволяющее быстро выполнять основные операции дерева поиска: добавление, удаление и поиск узла.

Сбалансированность достигается за счет введения дополнительного атрибута узла дерева - цвета.

При этом соблюдаются следующие требования:

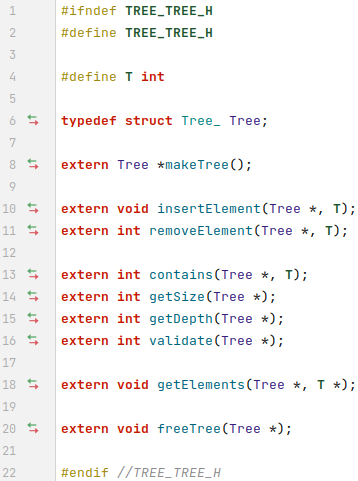
1. Корень всегда черный.
2. Все нулевые листья черные.
3. Оба потомка каждого красного узла черные.
4. Любой простой путь от узла-предка до листового узла-потомка содержит одинаковое число черных узлов.

Благодаря этим ограничениям, путь от корня до самого дальнего листа не более чем вдвое длиннее, чем до самого ближнего и дерево примерно сбалансировано.

После операции вставки/удаления требуется “починка” дерева через перекрас узлов и поворот поддеревьев.

# Описание API библиотеки

Интерфейс библиотеки состоит из следующих функций (файл **tree.h**):



* makeTree() - создает новое дерево с нулевым корнем и нулевым размером.
* insertElement() - вставка элемента. Если элемент уже есть в дереве ничего не меняется.
* removeElement() - удаление элемента. Если элемента не было в дереве ничего не меняется.
* contains() - проверяет, есть ли элемент в дереве.
* getSize() - получает количество элементов в дереве.
* getDepth() - получает максимальную глубину дерева.
* validate() - проверяет свойства дерева.
* getElements() - помещает в массив все элементы дерева от меньшего к большему.
* freeTree() - освобождает память из под дерева.

У реализации структуры **Tree** есть только два поля - корень и размер.

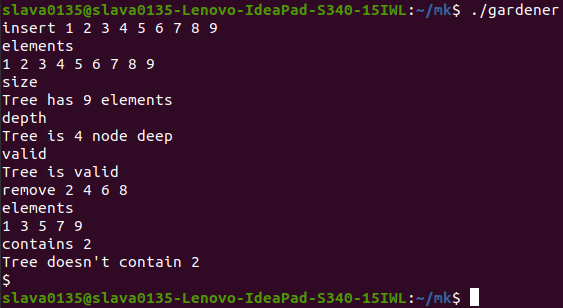
В целях абстракции используется тип T (определенный как int), который можно заменить на другой численный тип данных.

# Описание формата ввода

В качестве демонстрации работы структуры данных, была написана программа **gardener.c**. Она читает данные с **stdin** и отвечает в **stdout**. Операции задаются в следующем формате:

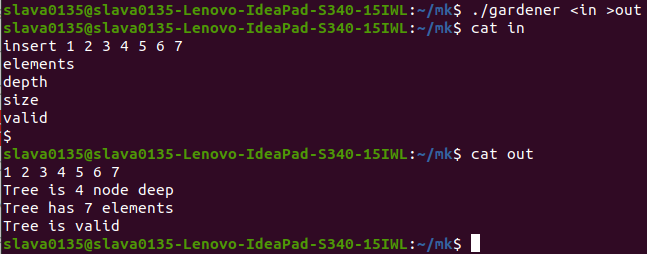
* **“insert [INT]...”** вставляет целые числа в дерево. **( insert 1 2 3 )**
* **“remove [INT]...”** удаляет целые числа из дерева. **( remove 1 2 3 )**
* **“contains [INT]”** проверяет входит ли элемент в дерево. ( **contains 1 )**
* **“elements”** печатает элементы дерева.
* **“size”** печатает размер дерева.
* **“depth”** печатает глубину дерева.
* **“valid”** проверяет валидность.

Для завершения работы нужно напечатать “$”.



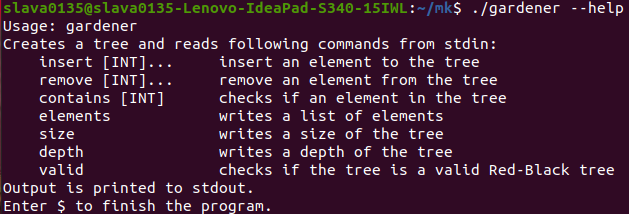
*Пример работы с программой*

Для ввода/вывода через файлы достаточно перенаправить **stdin** и **stdout**.



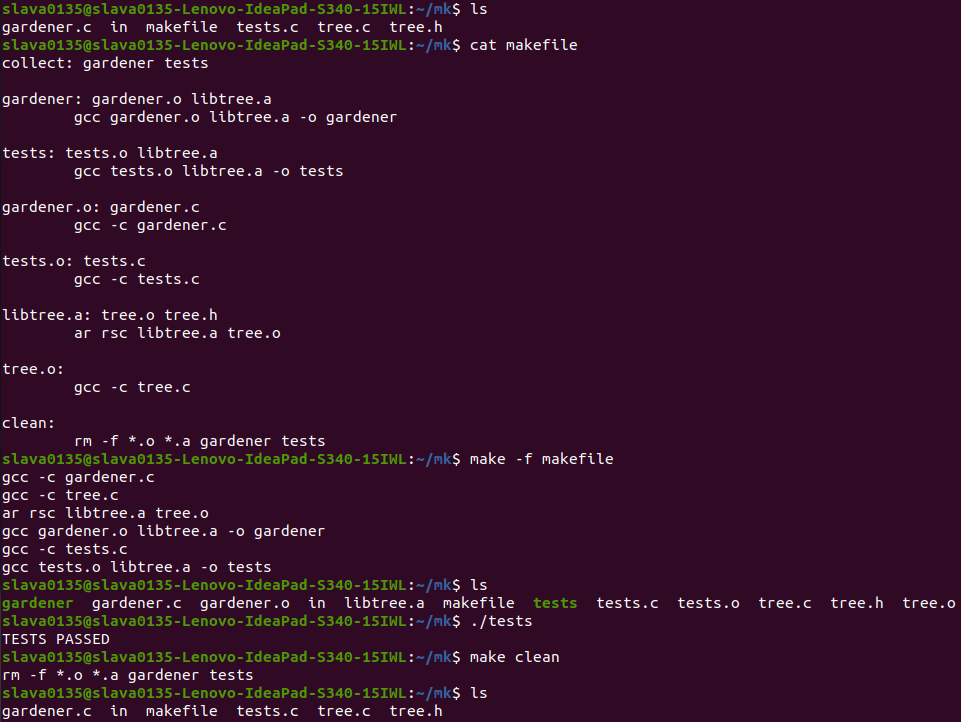
*Работа с файлами*

Вся информация о программе выводится по команде **gardener --help** (или **-h**)



# Сборка библиотеки и программ

Для осуществления сборки статической библиотеки и демонстрационной программы с модульными тестами было написан makefile.



*Сборка программы*

Сначала мы собираем статическую библиотеку, а потом используем ее при компиляции двух выходных программ (**gardener** и **tests**). Можно убедиться в правильности работы сборщика через запуск тестов (./tests).