

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Бинарные деревья

Студент гр. 9381

Матвеев А. Н.

Преподаватель

Фирсов М. А.

Санкт-Петербург

2020

Цель работы.

Познакомиться со структурой данных “бинарное дерево”, способами её организации и рекурсивной обработки. Получить навыки решения задач обработки бинарных деревьев.

Задание.

Вариант 9д

Рассматриваются бинарные деревья с элементами типа *Elem* (в качестве *Elem* использовать *char*). Заданы перечисления узлов некоторого дерева *b* в порядке КЛП и ЛКП. Требуется:

- восстановить дерево *b* и вывести его изображение;
- перечислить узлы дерева *b* в порядке ЛПК.

На базе указателей (динамическая связанная память).

Описание алгоритма.

Имеются два перечисления узлов бинарного дерева (далее - БД): КЛП и ЛКП. Они подаются в некоторую рекурсивную функцию R.

- 1) перечисление КЛП рассматривается поэлементно (берём по порядку каждый элемент этой последовательности, начиная с 1-го и до последнего).
- 2) Пусть каждый элемент КЛП - ключ, который ищется в строке ЛКП. После того, как ключ найден, текущая ЛКП-строка делится на 3 части: левое поддерево, корень и правое поддерево. За корень текущего узла принимается текущий ключ.
- 3) В каждом рекурсивном вызове имеются 2 пустых узла *p* и *q*, которые могут стать деревьями. На каждом вызове функции R узлу *p* присваивается возвращаемое значение данной функции, у которой на входе в качестве ЛКП-строки является левое поддерево текущей ЛКП-строки, а в качестве ключа - следующий за текущим в строке КЛП. Узлу *q* на каждом вызове функции R присваивается значение функции R, у которой на входе в качестве ЛКП-строки является правое поддерево текущей ЛКП строки, а в качестве ключа является следующий после

последнего обработанного (учитывается совершённый обход левого поддерева, сдвиг на 1 вперёд в КЛП-записи происходит при каждом рекурсивном вызове).

- 4) После того, как оба узла *p* и *q* инициализированы, создаётся новый узел, корнем которого является текущий ключ, а левым и правым поддеревьями - *p* и *q* соответственно. Данная функция *R* возвращает этот узел.
- 5) Условие конца алгоритма для конкретного узла - текущая лкп-строка пустая. В этом случае *R* возвращает *null*. Когда все ключи просмотрены, имеется уже построенное БД.

Выполнение работы.

Реализован класс *BinTree*, описывающий бинарное дерево.

Приватные поля:

- *Elem info = 0;* - данные узла
- *BinTree * leftTree;* - левое поддерево
- *BinTree * rightTree;* - правое поддерево

В качестве публичных методов класса имеются геттеры и сеттеры для корня, левого и правого поддерева, а также статические методы, не использующие экземпляры класса напрямую.

Рассмотрим их:

BinTree * getLeft() - возвращает указатель на левое поддерево бинарного дерева.

BinTree * getRight() - возвращает указатель на правое поддерево бинарного дерева.

[[nodiscard]] Elem rootBt() const - возвращает поле *info* текущего узла (тип *Elem*).

void setLeft(BinTree *pointer) - присваивает полю *leftTree* указатель на узел. Принимает на вход указатель на бинарное дерево *pointer*.

void setRight(BinTree *pointer) - присваивает полю *rightTree* указатель на узел. Принимает на вход указатель на бинарное дерево *pointer*.

void setRoot(Elem elem) - инициализирует поле info элементом типа elem.

Принимает на вход элемент шаблонного типа Elem.

~BinTree() = default; -деструктор класса BinTree.

BinTree() - конструктор класса BinTree. Инициализирует leftTree и rightTree нулевыми указателями.

static void destroy (BinTree * node) - рекурсивно удаляет выделенную под узлы память; Принимает указатель на узел.

static bool isNull(BinTree * node) - функция, проверяющая узел на null. Возвращает соответствующее логическое значение. Принимает указатель на узел.

static BinTree * consBT(const Elem &x, BinTree<char> *left, BinTree<char> *right) - функция-конструктор, выделяющая память под новый узел дерева и инициализирующая его поля. Принимает на вход указатели на левое и правое поддерево, а также данные, которые будет хранить этот узел. Возвращает указатель на созданный узел.

Все следующие функции имеют в качестве одного из аргументов ссылку на поток вывода out. Это необходимо, чтобы можно было выводить информацию как в консоль, так и в файл.

static void printLPK (BinTree<Elem>* node, std::ostream & out) - печатает дерево в лпк-формате. Принимает на вход указатель на дерево и ссылку на поток вывода out. Если узел непустой, то эта функция вызывается рекурсивно для левого поддерева, затем для правого поддерева, затем печатается корень.

static void printPKL (BinTree<Elem> *node, int move, std::ostream & out) выводит дерево справа налево с отступами. Принимает на вход указатель на узел, целочисленную переменную move (которая отвечает за количество отступов), а также ссылку на поток вывода. Эта функция позволяет наглядно представить бинарное дерево. Если текущий узел непуст, данная функция вызывается рекурсивно для правого поддерева, затем выводится корень с количеством отступов move (отступ для наглядности выглядит как “_”), после

чего происходит рекурсивный вызов для левого поддерева. При каждом рекурсивном вызове в качестве второго аргумента передаётся `move + 2`.

`static BinTree<Elem> * restoreBT(std::string &formatKLP, std::string &formatLKP, int &move, std::ostream & out, int n)` - ключевая функция в программе, реализующая описанный выше алгоритм. Позволяет восстановить бинарное дерево по КЛП- и ЛКП-перечислениям. Принимает на вход ссылки на строки `formatKLP` и `formatLKP`, ссылку на целочисленную переменную `move` (которая будет являться счётчиком при обходе строки KLP), целочисленное значение `n`, которое будет отвечать за количество отступов при отладочных выводах и ссылку на поток вывода `out`.

В самом начале программы делается проверка на пустоту строки ЛКП. Если она пуста, возвращается `nullptr`. Также сделаны аварийные проверки на то, что `move` меньше длины строки `formatKLP` и что ключ обязательно найдётся в текущей лкп-строке. В противном случае выводится сообщение о некоректности перечислений и программа аварийно завершается. Согласно алгоритму, при каждом запуске `restoreBT` создаётся переменная ключ, которая ищется в лкп-записи. Также созданы два указателя на БД `rightNode` и `leftNode`. Это будущие поддерева текущего узла. При помощи методов обработки строк типа `string` выделяются в текущей лкп-записи строки `left` и `right`, отображающие левые и правые поддерева для текущего узла.

Если `left` не пуста, то переменной `leftNode` присваивается значение, возвращаемое функцией `restoreBT` (происходит рекурсивный вызов), в качестве лкп-строки которой будет `left`, а индекс ключа будет увеличен на 1. Аналогично для `right` и `rightNode`. В этом случае индекс ключа тоже увеличится. При каждом рекурсивном вызове переменная `move` увеличивается на 1.

Функция возвращает конструктор `consBT`, в который в качестве корня передаётся `key`, а в качестве левого и правого узлов передаются `leftNode` и `rightNode` соответственно. Таким образом, для узлов динамически выделяется память, они инициализируются и восстанавливается БД.

static void printIndent(int n, std::ostream & out) - вспомогательная функция, которая выводит необходимое количество отступов в отладочных выводах. n - количество отступов в текущем выводе, out - ссылка на поток вывода.

Стоит отметить, что в классе BinTree реализовано шаблонное поле info типа Elem. В данной реализации программы в качестве шаблонного типа везде поставлен тип char.

В функции main() пользователю предоставлен выбор вводить данные как с консоли, так и из файла, выводить как в консоль, так и в файл. Обработаны возможные ошибки, которые может допустить пользователь при вводе. Учтено также, что в клп и лкп-представлениях в строках должно быть одинаковое количество символов и должен быть один и тот же набор символов. Прописаны все необходимые отладочные выводы. После того, как все необходимые данные введены, к ранее созданному указателю на экземпляр класса БД присваивается значение, возвращаемое функцией restoreBT, после чего запускается функция printPKL, восстановленное дерево выводится на экран, после чего запускается функция printLPK (вывод лпк-записи). После использования при помощи функции destroy память, выделенная под БД, очищается.

Исходный код программы смотреть в Приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	a	КЛП-запись: (a) ЛКП-запись: (a) Текущий ключ: (a), его номер в КЛП-строке: (1), в текущей ЛКП-строке: (1) Левое поддерев: (), корень: (a), правое поддерев: () Анализ левого поддерева завершён Анализ правого поддерева завершён Создание узла

		<p>Восстановленное дерево:</p> <p>_a</p> <p>Запись в ЛПК-виде:</p> <p>(a)</p> <p>Восстановленное дерево:</p> <p>_a</p> <p>Запись в ЛПК-виде:</p> <p>(a)</p>
2.	<p>abcde</p> <p>cbade</p>	<p>КЛП-запись: (abcde)</p> <p>ЛКП-запись: (cbade)</p> <p>Текущий ключ: (a), его номер в КЛП-строке: (1), в текущей ЛКП-строке: (3)</p> <p>Левое поддереву: (cb), корень: (a), правое поддереву: (de)</p> <p>Анализ левого поддереву</p> <p style="padding-left: 40px;">Текущий ключ: (b), его номер в КЛП-строке: (2), в текущей ЛКП-строке: (2)</p> <p style="padding-left: 40px;">Левое поддереву: (c), корень: (b), правое поддереву: ()</p> <p style="padding-left: 40px;">Анализ левого поддереву</p> <p style="padding-left: 80px;">Текущий ключ: (c), его номер в КЛП-строке: (3), в текущей ЛКП-строке: (1)</p> <p style="padding-left: 80px;">Левое поддереву: (), корень: (c), правое поддереву: ()</p> <p style="padding-left: 40px;">Анализ левого поддереву завершён</p> <p style="padding-left: 40px;">Анализ правого поддереву завершён</p> <p style="padding-left: 40px;">Создание узла</p> <p style="padding-left: 40px;">Анализ левого поддереву завершён</p> <p style="padding-left: 40px;">Анализ правого поддереву завершён</p> <p style="padding-left: 40px;">Создание узла</p>

		<p>Анализ левого поддерева завершён</p> <p>Анализ правого поддерева</p> <p>Текущий ключ: (d), его номер в КЛП-строке: (4), в текущей ЛКП-строке: (1)</p> <p>Левое поддерево: (), корень: (d), правое поддерево: (e)</p> <p>Анализ левого поддерева завершён</p> <p>Анализ правого поддерева</p> <p>Текущий ключ: (e), его номер в КЛП-строке: (5), в текущей ЛКП-строке: (1)</p> <p>Левое поддерево: (), корень: (e), правое поддерево: ()</p> <p>Анализ левого поддерева завершён</p> <p>Анализ правого поддерева завершён</p> <p>Создание узла</p> <p>Анализ правого поддерева завершён</p> <p>Создание узла</p> <p>Анализ правого поддерева завершён</p> <p>Создание узла</p> <p>Восстановленное дерево:</p> <pre> _____e ____d _a ____b _____c </pre> <p>Запись в ЛПК-виде:</p> <p>(cbeda)</p>
3.	abdecfgh dbeacgfh	<p>КЛП-запись: (abdecfgh)</p> <p>ЛКП-запись: (dbeacgfh)</p> <p>Текущий ключ: (a), его номер в КЛП-строке: (1), в текущей ЛКП-строке: (4)</p>

		<p>Левое поддерево: (dbe), корень: (a), правое поддерево: (cgfh)</p> <p>Анализ левого поддерева</p> <p>Текущий ключ: (b), его номер в КЛП-строке: (2), в текущей ЛКП-строке: (2)</p> <p>Левое поддерево: (d), корень: (b), правое поддерево: (e)</p> <p>Анализ левого поддерева</p> <p>Текущий ключ: (d), его номер в КЛП-строке: (3), в текущей ЛКП-строке: (1)</p> <p>Левое поддерево: (), корень: (d), правое поддерево: ()</p> <p>Анализ левого поддерева завершён</p> <p>Анализ правого поддерева завершён</p> <p>Создание узла</p> <p>Анализ левого поддерева завершён</p> <p>Анализ правого поддерева</p> <p>Текущий ключ: (e), его номер в КЛП-строке: (4), в текущей ЛКП-строке: (1)</p> <p>Левое поддерево: (), корень: (e), правое поддерево: ()</p> <p>Анализ левого поддерева завершён</p> <p>Анализ правого поддерева завершён</p> <p>Создание узла</p> <p>Анализ правого поддерева завершён</p> <p>Создание узла</p> <p>Анализ левого поддерева завершён</p> <p>Анализ правого поддерева</p> <p>Текущий ключ: (c), его номер в КЛП-строке: (5), в текущей ЛКП-строке: (1)</p> <p>Левое поддерево: (), корень: (c), правое поддерево: (gfh)</p> <p>Анализ левого поддерева завершён</p> <p>Анализ правого поддерева</p>
--	--	---

		<p>Текущий ключ: (f), его номер в КЛП-строке: (6), в текущей ЛКП-строке: (2)</p> <p>Левое поддерево: (g), корень: (f), правое поддерево: (h)</p> <p>Анализ левого поддерева</p> <p>Текущий ключ: (g), его номер в КЛП-строке: (7), в текущей ЛКП-строке: (1)</p> <p>Левое поддерево: (), корень: (g), правое поддерево: ()</p> <p>Анализ левого поддерева завершён</p> <p>Анализ правого поддерева завершён</p> <p>Создание узла</p> <p>Анализ левого поддерева завершён</p> <p>Анализ правого поддерева</p> <p>Текущий ключ: (h), его номер в КЛП-строке: (8), в текущей ЛКП-строке: (1)</p> <p>Левое поддерево: (), корень: (h), правое поддерево: ()</p> <p>Анализ левого поддерева завершён</p> <p>Анализ правого поддерева завершён</p> <p>Создание узла</p> <p>Анализ правого поддерева завершён</p> <p>Создание узла</p> <p>Анализ правого поддерева завершён</p> <p>Создание узла</p> <p>Анализ правого поддерева завершён</p> <p>Создание узла</p> <p>Восстановленное дерево:</p> <pre> _____h _____f _____g _____c </pre>
--	--	---

		<p>_a</p> <p>_____e</p> <p>____b</p> <p>_____d</p> <p>Запись в ЛПК-виде: (debghfca)</p>
4.	<p>abcde</p> <p>cbade</p>	<p>КЛП-запись: (abcde)</p> <p>ЛКП-запись: (cbade)</p> <p>Текущий ключ: (a), его номер в КЛП-строке: (1), в текущей ЛКП-строке: (3)</p> <p>Левое поддерево: (cb), корень: (a), правое поддерево: (de)</p> <p>Анализ левого поддерева</p> <p style="padding-left: 40px;">Текущий ключ: (b), его номер в КЛП-строке: (2), в текущей ЛКП-строке: (2)</p> <p style="padding-left: 40px;">Левое поддерево: (c), корень: (b), правое поддерево: ()</p> <p style="padding-left: 40px;">Анализ левого поддерева</p> <p style="padding-left: 80px;">Текущий ключ: (c), его номер в КЛП-строке: (3), в текущей ЛКП-строке: (1)</p> <p style="padding-left: 80px;">Левое поддерево: (), корень: (c), правое поддерево: ()</p> <p style="padding-left: 40px;">Анализ левого поддерева завершён</p> <p style="padding-left: 40px;">Анализ правого поддерева завершён</p> <p style="padding-left: 40px;">Создание узла</p> <p style="padding-left: 40px;">Анализ левого поддерева завершён</p> <p style="padding-left: 40px;">Анализ правого поддерева завершён</p> <p style="padding-left: 40px;">Создание узла</p> <p>Анализ левого поддерева завершён</p> <p>Анализ правого поддерева</p> <p style="padding-left: 40px;">Текущий ключ: (d), его номер в КЛП-строке: (4), в текущей ЛКП-строке: (1)</p>

		<p>Левое поддерево: (), корень: (d), правое поддерево: (e)</p> <p>Анализ левого поддерева завершён</p> <p>Анализ правого поддерева</p> <p>Текущий ключ: (e), его номер в КЛП-строке: (5), в текущей ЛКП-строке: (1)</p> <p>Левое поддерево: (), корень: (e), правое поддерево: ()</p> <p>Анализ левого поддерева завершён</p> <p>Анализ правого поддерева завершён</p> <p>Создание узла</p> <p>Анализ правого поддерева завершён</p> <p>Создание узла</p> <p>Анализ правого поддерева завершён</p> <p>Создание узла</p> <p>Восстановленное дерево:</p> <pre> _____e ____d _a ____b _____c </pre> <p>Запись в ЛПК-виде:</p> <p>(cbeda)</p>
5.	<p>abcedff</p> <p>cebfdaf</p>	<p>КЛП-запись: (abcedff)</p> <p>ЛКП-запись: (cebfdaf)</p> <p>Текущий ключ: (a), его номер в КЛП-строке: (1), в текущей ЛКП-строке: (6)</p> <p>Левое поддерево: (cebfd), корень: (a), правое поддерево: (f)</p> <p>Анализ левого поддерева</p> <p>Текущий ключ: (b), его номер в КЛП-строке: (2), в текущей ЛКП-строке: (3)</p>

		<p>Левое поддереву: (се), корень: (b), правое поддереву: (fd)</p> <p>Анализ левого поддереву</p> <p>Текущий ключ: (с), его номер в КЛП-строке: (3), в текущей ЛКП-строке: (1)</p> <p>Левое поддереву: (), корень: (с), правое поддереву: (е)</p> <p>Анализ левого поддереву завершён</p> <p>Анализ правого поддереву</p> <p>Текущий ключ: (е), его номер в КЛП-строке: (4), в текущей ЛКП-строке: (1)</p> <p>Левое поддереву: (), корень: (е), правое поддереву: ()</p> <p>Анализ левого поддереву завершён</p> <p>Анализ правого поддереву завершён</p> <p>Создание узла</p> <p>Анализ правого поддереву завершён</p> <p>Создание узла</p> <p>Анализ левого поддереву завершён</p> <p>Анализ правого поддереву</p> <p>Текущий ключ: (d), его номер в КЛП-строке: (5), в текущей ЛКП-строке: (2)</p> <p>Левое поддереву: (f), корень: (d), правое поддереву: ()</p> <p>Анализ левого поддереву</p> <p>Текущий ключ: (f), его номер в КЛП-строке: (6), в текущей ЛКП-строке: (1)</p> <p>Левое поддереву: (), корень: (f), правое поддереву: ()</p> <p>Анализ левого поддереву завершён</p> <p>Анализ правого поддереву завершён</p> <p>Создание узла</p> <p>Анализ левого поддереву завершён</p>
--	--	---

		<p>Анализ правого поддерева завершён</p> <p>Создание узла</p> <p>Анализ правого поддерева завершён</p> <p>Создание узла</p> <p>Анализ левого поддерева завершён</p> <p>Анализ правого поддерева</p> <p>Текущий ключ: (f), его номер в КЛП-строке: (6), в текущей ЛКП-строке: (1)</p> <p>Левое поддерево: (), корень: (f), правое поддерево: ()</p> <p>Анализ левого поддерева завершён</p> <p>Анализ правого поддерева завершён</p> <p>Создание узла</p> <p>Анализ правого поддерева завершён</p> <p>Создание узла</p> <p>Восстановленное дерево:</p> <pre> f / a / \ d f / \ / b e c </pre> <p>Запись в ЛПК-виде: (ecfdbfa)</p>
--	--	---

Тестирование на некорректных данных.

Результаты тестирования на некорректных данных представлены в табл.

2.

Таблица 2 - результаты тестирования на некорректных данных

№ п/п	Входные данные	Выходные данные
-------	----------------	-----------------

1.	igrigerig gnergnenk	ОШИБКА! Символы в строках должны быть одни и те же
2.	hjietjh ghj	ОШИБКА! Количество элементов должно совпадать
3.	gjkepg kgjepg	КЛП-запись: (gjkepg) ЛКП-запись: (kgjepg) Текущий ключ: (g), его номер в КЛП-строке: (1), в текущей ЛКП-строке: (2) Левое поддереву: (k), корень: (g), правое поддереву: (jepg) Анализ левого поддереву Ключ (j) не найден в ЛКП-подстроке (k). По данной записи невозможно восстановить бинарное дерево. Проверьте корректность перечислений

Выводы.

Ознакомился со структурой данных “бинарное дерево”, способами её организации и рекурсивной обработки. Получил навыки решения задач обработки бинарных деревьев.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp:

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <cstdlib>
#include <string>
using namespace std;

template <typename Elem>
class BinTree{// бинарное дерево
private:
    Elem info = 0; // данные узла
    BinTree * leftTree; // левое поддерев
    BinTree * rightTree; // правое поддерев
public:
    BinTree() {
        leftTree = nullptr;
        rightTree = nullptr;
    }// конструктор класса БД

    // геттеры
    BinTree * getLeft(){
        return this->leftTree;
    }
    BinTree * getRight(){
        return this->rightTree;
    }

    [[nodiscard]] Elem rootBT() const{
        return this->info;
    }

    // сеттеры
    void setLeft(BinTree *pointer){
        this->leftTree = pointer;
    }
}
```



```

void setRight(BinTree *pointer){
    this->rightTree = pointer;
}
void setRoot(Elem elem){
    this->info = elem;
}

~BinTree() = default; // деструктор класса БД

static bool isNull(BinTree * node){ // проверка на пустой узел
    return (node == nullptr);
}

static void destroy (BinTree * node) // рекурсивно удаляет
выделенную под узлы память
{
    if (!isNull(node)){
        destroy (node->getLeft());
        destroy (node->getRight());
        delete node;
        node = nullptr;
    }
}

static BinTree * consBT(const Elem &x, BinTree<char> *left,
BinTree<char> *right)// конструктор БД (создаёт элемент бинарного
дерева)
{
    auto* p = new BinTree<char>;
    if (!isNull(p)) {
        p->setRoot(x);
        p->setLeft(left);
        p->setRight(right);
        return p;
    }
    else {
        std::cerr << "Memory not enough\n";
        exit(0);
    }
}

```

```

        static BinTree<Elem> * restoreBT(std::string &formatKLP,
std::string &formatLKP, int &move, std::ostream & out, int n){
    // функция восстанавливающая БД по КЛП И ЛКП записи

    if(move >= formatKLP.size()){
        out << "Невозможно восстановить дерево. Проверьте
корректность перечислений\n";
        exit(0);
    }

    char key = formatKLP[move]; // клп-запись - это ориентир для
лкп-записи

    if(formatLKP.find(key) == -1){
        out << "Ключ (" << key << ") не найден в ЛКП-подстроке ("
<< formatLKP << ").\n"
        <<"По данной записи невозможно восстановить бинарное
дерево. Проверьте корректность перечислений\n";
        exit(0);
    }
    //условие конца рекурсии
    if(formatLKP == ""){
        return nullptr;
    }

    BinTree <Elem>* leftNode = nullptr;
    BinTree<Elem> * rightNode = nullptr;
    printIndent(n, out);
    out << "Текущий ключ: (" << key << "), его номер в КЛП-строке:
(" << formatKLP.find(key) + 1 << "), в текущей ЛКП-строке: (" <<
formatLKP.find(key) + 1 << ")\n";
    std::string left = formatLKP.substr(0, formatLKP.find(key) );
    //выделение левого поддеревя ( всё что слева от корня )
    std::string right = formatLKP.substr(formatLKP.find(key) + 1);
    //выделение левого поддеревя ( всё что справа от корня )
    printIndent(n, out);
    out << "Левое поддерево: (" << left << "), корень: (" << key
<< "), правое поддерево: (" << right << ")\n";
    if(!left.empty()) {

```

```

        printIndent(n, out);
        out << "Анализ левого поддерева\n";
        leftNode = restoreBT(formatKLP, left, ++move, out, n + 1);
// если левая часть лкп не пуста, создаётся левое поддерево
    }
    printIndent(n, out);
    out << "Анализ левого поддерева завершён\n";
    if(!right.empty()) {
        printIndent(n, out);
        out << "Анализ правого поддерева\n";
        rightNode = restoreBT(formatKLP, right, ++move, out, n +
1); // если правая часть лкп не пуста, создаётся правое поддерево
    }
    printIndent(n, out);
    out << "Анализ правого поддерева завершён\n";

    printIndent(n, out);
    out << "Создание узла\n";
    return BinTree<Elem>::consBT(key, leftNode, rightNode); //
генерация узла
    }

    static void printLPK (BinTree<Elem>* node, std::ostream & out) //
вывод в лпк-формате
    {
        if (!BinTree<Elem>::isNull(node)) {
            printLPK (node->getLeft(), out);
            printLPK (node->getRight(), out);
            out << node->rootBT();
        }
    }

    static void printPKL (BinTree<Elem> *node, int move, std::ostream &
out) // функция, изображающая бинарное дерево (повернутое) в удобном
формате
    { // БД выводится справа налево с необходимым количеством отступов
        // это позволяет визуализировать БД в привычном виде
        if (!BinTree<Elem>::isNull(node)) {
            printPKL (node->getRight(), move + 2, out);

```

```

        for (int i = 0; i < move; i++)
            out << "_";
        out << node->rootBT() << std::endl;
        printPKL (node->getLeft(), move + 2, out);
    }
}

static void printIndent(int n, std::ostream & out){
    for(int i = 0; i < n; i++ )
        out << "\t";
}

};

namespace file{
    ifstream inFile;
    ofstream outFile;
}

int main(){
    char flag;
    string path;
    BinTree <char> * tree; // корень БД
    string formatKLP;
    string formatLKP;
    int keyMover = 0;
    int n = 0;
    char info;
    cout << "0 - ввод с консоли\n1 - ввод с файла\n";
    cin >> flag;
    switch(flag){
        case '0':
            cout << "Введите сначала в КЛП-формате, затем перейдите на
следующую строку введите в ЛКП-формате:\n";
            cin >> formatKLP; //ввод КЛП-записи
            cin >> formatLKP; // ввод ЛКП-записи
            break;

```

```

    case '1':
        cout << "Введите путь до файла:\n";
        cin >> path;
        file::inFile.open(path);
        if(!file::inFile.is_open()){
            cout << "Невозможно открыть файл на чтение\n";
            return 0;
        }
        file::inFile >> formatKLP;
        file::inFile >> formatLKP;
        file::inFile.close();
        break;
    default:
        cout << "Неверный формат\n";
        return 0;
}
// отлов ошибок

cout << "0 - вывод в консоль\n1 - вывод в файл\n";
cin >> flag;
switch(flag) {
    case '0':
        break;
    case '1':
        cout << "Введите путь до файла:\n";
        cin >> path;
        file::outFile.open(path);
        if(!file::outFile.is_open()){ // обработка ошибочных данных
            cout << "Невозможно открыть файл на запись\n";
            return 0;
        }
        break;
    default:
        cout << "Неверный формат\n";
        return 0;
}

ostream & out = flag == '0' ? cout : file::outFile; // установка
потока вывода
if(formatKLP.size() != formatLKP.size()){

```

```

        out << "ОШИБКА! Количество элементов должно совпадать\n";
        return 0;
    }

    for(char i : formatKLP){
        if(formatLKP.find(i) == -1){
            out << "ОШИБКА! Символы в строках должны быть одни и те
же\n";
            return 0;
        }
    }

    out << "КЛП-запись: (" << formatKLP << ")\n";
    out << "ЛКП-запись: (" << formatLKP << ")\n";
    tree = BinTree <char>::restoreBT(formatKLP, formatLKP, keyMover,
out, n); // восстановление БД по КЛП и ЛКП видам записи
    out << "\nВосстановленное дерево:\n";
    BinTree <char>::printPKL(tree, 1, out); // вывод на экран БД
    out << "\nЗапись в ЛПК-виде:\n(";
    BinTree <char>::printLPK(tree, out); // вывод в лпк формате
    out << ")\n";
    BinTree <char>::destroy(tree); // освобождение памяти
    file::outFile.close();
    return 0;
}

```