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

# ОТЧЕТ

**по лабораторной работе №4**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: Бинарные деревья**

Студентка гр. 8383 _____ Ишанина Л.Н.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

**Цель работы.**

Изучить структуру данных бинарное дерево. Научиться его создавать, обходить и выполнять некоторые операции с ним.

**Основные теоретические сведения**

*Дерево* – конечное множество $T$, состоящее из одного или более узлов, таких, что

а) имеется один специально обозначенный узел, называемый *корнем* данного дерева;

б) остальные узлы (исключая корень) содержатся в $m \varepsilon 0$ попарно не пересекающихся множествах $T_1, T_2, ..., T_m$, каждое из которых, в свою очередь, является деревом. Деревья $T_1, T_2, ..., T_m$ называются *поддеревьями* данного дерева.

При программировании и разработке вычислительных алгоритмов удобно использовать именно такое *рекурсивное* определение, поскольку рекурсивность является естественной характеристикой этой структуры данных.

Каждый узел дерева является корнем некоторого поддерева. В том случае, когда множество поддеревьев такого корня пусто, этот узел называется *концевым узлом*, или *листом*. *Уровень* узла определяется рекурсивно следующим образом: 1) корень имеет уровень 1; 2) другие узлы имеют уровень, на единицу больший их уровня в содержащем их поддереве этого корня. Используя для уровня узла $a$ дерева $T$ обозначение *уровень* $(a, T)$, можно записать это определение в виде

$$\textit{уровень}(a,T) = \begin{cases} 1, & \text{если } a - \text{корень дерева } T \\ \textit{уровень}(a,T_i)+1, & \text{если } a - \text{не корень дерева } T \end{cases}$$

где $T_i$ – поддерево корня дерева $T$, такое, что $a \square T_i$.

Говорят, что каждый корень является *отцом* корней своих поддеревьев и что последние являются *сыновьями* своего отца и *братьями* между собой. Говорят также, что узел $n$ – *предок* узла $m$ (а узел $m$ – *потомок* узла $n$), если $n$ – либо отец $m$, либо отец некоторого предка $m$.

Если в определении дерева существен порядок перечисления поддеревьев $T_1, T_2, ..., T_m$, то дерево называют *упорядоченным* и говорят о «первом» ($T_1$), «втором» ($T_2$) и т. д. поддеревьях данного корня. Далее будем считать, что все рассматриваемые деревья являются упорядоченными, если явно не оговорено противное. Отметим также, что в терминологии теории графов определенное ранее упорядоченное дерево более полно называлось бы «конечным ориентированным (корневым) упорядоченным деревом».

*Лес* – это множество (обычно упорядоченное), состоящее из некоторого (быть может, равного нулю) числа непересекающихся деревьев. Используя понятие леса, пункт *б* в определении дерева можно было бы сформулировать так: *узлы*

*дерева, за исключением корня, образуют лес*.

Рассмотрим функциональную спецификацию структуры данных дерева с узлами типа α: *Tree of* α = *Tree* (α). При этом лес деревьев *Forest* (α) определим как *L_list* (*Tree* (α)) через уже известную структуру линейного списка *L_list* с базовыми функциями *Cons*, *Head*, *Tail*, *Null* (см. 1.6). Базовые операции с деревом задаются набором функций:

1) *Root*: *Tree* → α;

2) *Listing*: *Tree* → *Forest*;

3) *ConsTree*: α → *Forest* → *Tree*

и аксиомами (*u*: α; *f*: *Forest* (α); *t*: *Tree* (α)):

A1) *Root* (*ConsTree* (*u*, *f*)) = *u*;

A2) *Listing* (*ConsTree* (*u*, *f*)) = *f*;

A3) *ConsTree* (*Root* (*t*), *Listing* (*t*)) = *t*.

Здесь функции *Root* и *Listing* селекторы: *Root* выделяет корень дерева, а *Listing* выделяет лес поддеревьев корня данного дерева. Конструктор *ConsTree* порождает дерево из заданных узла и леса деревьев.

Тот факт, что структура данных *Forest* явно определена через *L_list* (*Tree*), позволяет реализовать структуру дерева (леса) на базе другой структуры данных, а именно на базе иерархических списков. Достаточно рассматривать при этом описанное в 3.1 скобочное представление дерева как *S*-выражение специальной структуры. Возможно и другое удобное представление дерева (леса), основанное на некотором соответствии леса и бинарного дерева, описанном далее в 3.3.

Рассмотрим функциональную спецификацию структуры данных бинарного дерева с узлами типа α: *BinaryTree* (α) α *BT* (α). Здесь важно различать ситуации обработки пустого и непустого бинарных деревьев, поскольку некоторые операции определяются только на непустых бинарных деревьях. Далее считаем, что значение типа *BT* есть либо $\not\subset$ (пустое бинарное дерево), либо значение типа *NonNullBT*. Тогда базовые операции типа *BT* (α) задаются набором функций:

1) *Root*: *NonNullBT* → α;

2) *Left*: *NonNullBT* → *BT*;

3) *Right*: *NonNullBT* → *BT*;

4) *ConsBT*: α + *BT* + *BT* + *NonNullBT*;

5) *Null*: *BT* → *Boolean*;

6) $\not\subset$: → *BT*

и набором аксиом (*u*: α, *b*: *NonNullBT* (α), *b*1, *b*2: *BT* (α)):

A1) *Null* ($\not\subset$) = *true*;

A1') *Null* (*b*) = *false*;

A2) *Null* (*ConsBT* (*u*, *b*1, *b*2)) = *false*;

A3) *Root* (*ConsBT* (*u*, *b*1, *b*2)) = *u*;

A4) *Left* (*ConsBT* (*u*, *b*1, *b*2)) = *b*1;

A5) *Right* (*ConsBT* (*u*, *b*1, *b*2)) = *b*2;

А6) *ConsBT* (*Root* (*b*), *Left* (*b*), *Right* (*b*)) = *b*.

Здесь функции *Root*, *Left* и *Right* селекторы: *Root* выделяет корень бинарного дерева, а *Left* и *Right* его левое и правое поддеревья соответственно. Конструктор *ConsBT* порождает бинарное дерево из заданных узла и двух бинарных деревьев. Предикат *Null* индикатор, различающий пустое и непустое бинарные деревья.

**Постановка задачи.**

**Вариант 9-д**.

Рассматриваются бинарные деревья с элементами типа Elem. Заданы перечисления узлов некоторого дерева b либо в порядке ЛКП, либо в порядке ЛПК. Требуется:

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

**Описание структуры узла дерева.**

Структура узла бинарного дерева содержит в себе переменные: char info – содержит в себе информацию о том, что хранится в узле бинарного дерева, node* lt – указатель на левый узел бинарного дерева и node* rt – указатель на правый узел бинарного дерева.

**Описание алгоритмов и функций.**

Получение из строки, представленной в виде обхода ЛКП или ЛПК, бинарного дерева осуществляется с помощью двух функций: makeTreeLKP(str, i, tree, err, counter, t, fout) и makeTreeLPK(str, i, tree, err, counter, t, fout) соответственно.

Алгоритм рекурсивных функций makeTreeLKP(str, i, tree, err, counter, t, fout) и makeTreeLPK(str, i, tree, err, counter, t, fout) схож, и заключается в том, что проходясь по строке, функция поочередно записывает в каждый узел соответствующий ему элемент из строки, “привязывая” при этом с предыдущим корнем узла, согласно порядку обхода ЛКП или ЛПК.

Вывод бинарного дерева в порядке КЛП происходит с помощью функции void printKLP(binTree b, ofstream& fout).

Алгоритм данной функции заключается в выводе сначала корневого узла, затем, левого и правого.

И вывод самого бинарного дерева на экран, осуществляется с помощью функции void printTree(node* p, int level, ofstream& fout).
Алгоритм функции состоит из рекурсивного вызова функции сначала с указателем на правый узел, затем, отделяя табуляциями, выводится корень, и снова рекурсивно происходит вызов, но уже с указанием на левый узел.

Описание функции makeTreeLKP(str, i, tree, err, counter, t, fout).
Функция принимает на вход строку, индекс, узел бинарного дерева, переменную-флаг, сообщающую об ошибках, счетчик, необходимый для изображения на экране рекурсии, переменную-флаг, которая позволяет грамотно выводить описание промежуточных значений и переменную для файлового вывода. В начале, проверив не является ли строка пустой, создается узел бинарного дерева, увеличивается индекс (делаем шаг) строки и рекурсивно вызывается функция с указание на правый узел бинарного дерева, далее, записав в него значение из строки, и сделав проверки на наличие возможных ошибок, записывается элемент в корневой узел дерева, потом функция снова рекурсивно вызывает себя, но с указание на левый узел бинарного дерева, и записав в него значение элемента строки, возвращается и снова делает проверки на наличие возможных ошибок. Все промежуточные значения выводятся как на консоль, так и в файл.

Описание функции makeTreeLPK(str, i, tree, err, counter, t, fout).
Функция также принимает на вход строку, индекс, узел бинарного дерева, переменную-флаг, сообщающую об ошибках, счетчик, необходимый для изображения на экране рекурсии, переменную-флаг, которая позволяет грамотно выводить описание промежуточных значений и переменную для файлового вывода. В начале, проверив не является ли строка пустой, создается узел бинарного дерева, увеличивается индекс (делаем шаг) строки и рекурсивно вызывается функция с указание на правый узел бинарного дерева, далее, записав в него значение из строки, и сделав проверки на наличие возможных

ошибок, функция снова рекурсивно вызывает себя, но с указание на левый узел бинарного дерева, и записав в него значение элемента строки, возвращается и снова делает проверки на наличие возможных ошибок. Далее записывается элемент в корневой узел дерева. Все промежуточные значения выводятся как на консоль, так и в файл.

Описание функции printKLP(binTree b, ofstream& fout).
На вход функция принимает дерево и переменную, необходимую для вывода в файл. Далее происходит проверка, на “непустое” дерево с помощью функции bool isNull(binTree b), и в случае отрицательного возврата выводится содержание корня дерева. Затем, функция рекурсивно вызывает себя и то же самое проделывает с левым узлом, а затем и с правым. Все промежуточные значения выводятся как на консоль, так и в файл.

Описание функции void printTree(binTree p, int level, ofstream& fout).
Функция принимает на вход дерево, переменную-счетчик, необходимую для расстановки табуляции в зависимости от уровня дерева и переменную для файлового ввода. Сначала функция рекурсивно вызывает себя, с указанием на правый узел дерева, при этом увеличивая переменную счетчик, затем расставляет табуляции в зависимости от этой переменной, выводит на экран информацию узла дерева, и рекурсивно вызывает себя с указание на левый узел дерева. Все промежуточные значения выводятся как на консоль, так и в файл.
Описание функции main().
В данной функции происходит считывание из файла или считывание с консоли, в зависимости от выбора пользователя, вызов либо рекурсивной функции makeTreeLKP(), получающей из строки (порядок обхода ЛКП) бинарное дерево, либо makeTreeLPK(), получающей из строки (порядок обхода ЛПК) бинарное дерево, по результатам одной из вышеперечисленных функций выводится порядок КЛП-обхода бинарного дерева и само изображение дерева на экран. Все промежуточные значения выводятся как на консоль, так и в файл.

**Тестирование.**

Тесты на неверных данных и крайних значениях:

Тест1 – порядок ЛКП: (12)

```
Выберите действие:
0 - выход
1 - ввод с файла в порядке ЛКП
2 - ввод с файла в порядке ЛПК
3 - ввод с консоли в порядке ЛКП
4 - ввод с консоли в порядке ЛПК
3
Пожалуйста, введите выражение.
(12)
Запускается функция makeTreeLKP!
 Осуществляется проход по строке. На данный момент находимся:   (
  Осуществляется проход по строке. На данный момент находимся:   1
  Записываем символ в левый узел дерева!!!
 Осуществляется проход по строке. На данный момент находимся:   2
 Записываем символ в корневой узел дерева!!!
Ошибка! после корневого узла не введен правый узел!
Функция makeTreeLKP завершает работу!
```

Тест2 – порядок ЛКП: ((1#2)+(3*4))

```
Выберите действие:
0 - выход
1 - ввод с файла в порядке ЛКП
2 - ввод с файла в порядке ЛПК
3 - ввод с консоли в порядке ЛКП
4 - ввод с консоли в порядке ЛПК
3
Пожалуйста, введите выражение.
((1#2)+(3*4))
Запускается функция makeTreeLKP!
 Осуществляется проход по строке. На данный момент находимся:   (
  Осуществляется проход по строке. На данный момент находимся:   (
   Осуществляется проход по строке. На данный момент находимся:   1
   Записываем символ в левый узел дерева!!!
Ошибка! Пустая голова!
```

Тест3 – порядок ЛПК: (a)

```
Выберите действие:
0 - выход
1 - ввод с файла в порядке ЛКП
2 - ввод с файла в порядке ЛПК
3 - ввод с консоли в порядке ЛКП
4 - ввод с консоли в порядке ЛПК
4
Пожалуйста, введите выражение.
(a)
Запускается функция makeTreeLPK!
 Осуществляется проход по строке. На данный момент находимся:   (
  Осуществляется проход по строке. На данный момент находимся:   a
  Записываем символ в левый узел дерева!!!
Ошибка! после левого узла не введен правый, либо его пустое обозначение!
Функция makeTreeLPK завершает работу!
```

Тест4 – порядок ЛПК: ((((12-)(34+)*)((56+)(7*)-)+)

```
0 - выход
1 - ввод с файла в порядке ЛКП
2 - ввод с файла в порядке ЛПК
3 - ввод с консоли в порядке ЛКП
4 - ввод с консоли в порядке ЛПК
4
Пожалуйста, введите выражение.
(((12-)(34+)*)((56+)(7*)-)+)
Запускается функция makeTreeLPK!
 Осуществляется проход по строке. На данный момент находимся:   (
  Осуществляется проход по строке. На данный момент находимся:   (
   Осуществляется проход по строке. На данный момент находимся:   (
    Осуществляется проход по строке. На данный момент находимся:   1
    Записываем символ в левый узел дерева!!!
    Осуществляется проход по строке. На данный момент находимся:   2
    Записываем символ в правый узел дерева!!!
   Осуществляется проход по строке. На данный момент находимся:   -
   Записываем символ в корневой узел дерева!!!
   Осуществляется проход по строке. На данный момент находимся:   )
   Осуществляется проход по строке. На данный момент находимся:   (
    Осуществляется проход по строке. На данный момент находимся:   3
    Записываем символ в левый узел дерева!!!
    Осуществляется проход по строке. На данный момент находимся:   4
    Записываем символ в правый узел дерева!!!
   Осуществляется проход по строке. На данный момент находимся:   +
   Записываем символ в корневой узел дерева!!!
   Осуществляется проход по строке. На данный момент находимся:   )
  Осуществляется проход по строке. На данный момент находимся:   *
  Записываем символ в корневой узел дерева!!!
```

```
  Осуществляется проход по строке. На данный момент находимся:   )
  Осуществляется проход по строке. На данный момент находимся:   (
   Осуществляется проход по строке. На данный момент находимся:   (
    Осуществляется проход по строке. На данный момент находимся:   5
    Записываем символ в левый узел дерева!!!
    Осуществляется проход по строке. На данный момент находимся:   6
    Записываем символ в правый узел дерева!!!
   Осуществляется проход по строке. На данный момент находимся:   +
   Записываем символ в корневой узел дерева!!!
   Осуществляется проход по строке. На данный момент находимся:   )
   Осуществляется проход по строке. На данный момент находимся:   (
    Осуществляется проход по строке. На данный момент находимся:   7
    Записываем символ в левый узел дерева!!!
    Осуществляется проход по строке. На данный момент находимся:   *
    Записываем символ в правый узел дерева!!!
Ошибка! после правого узла не введен корень!
```

Верные тесты:

Тест 5 – порядок ЛКП: (((1-2)*(3+4))+((5+6)-(7*8)))

```
Выберите действие:
0 - выход
1 - ввод с файла в порядке ЛКП
2 - ввод с файла в порядке ЛПК
3 - ввод с консоли в порядке ЛКП
4 - ввод с консоли в порядке ЛПК
3
Пожалуйста, введите выражение.
(((1-2)*(3+4))+((5+6)-(7*8)))
Запускается функция makeTreeLKP!
 Осуществляется проход по строке. На данный момент находимся:   (
  Осуществляется проход по строке. На данный момент находимся:   (
   Осуществляется проход по строке. На данный момент находимся:   (
    Осуществляется проход по строке. На данный момент находимся:   1
    Записываем символ в левый узел дерева!!!
   Осуществляется проход по строке. На данный момент находимся:   -
   Записываем символ в корневой узел дерева!!!
    Осуществляется проход по строке. На данный момент находимся:   2
    Записываем символ в правый узел дерева!!!
   Осуществляется проход по строке. На данный момент находимся:   )
  Осуществляется проход по строке. На данный момент находимся:   *
  Записываем символ в корневой узел дерева!!!
   Осуществляется проход по строке. На данный момент находимся:   (
    Осуществляется проход по строке. На данный момент находимся:   3
    Записываем символ в левый узел дерева!!!
   Осуществляется проход по строке. На данный момент находимся:   +
   Записываем символ в корневой узел дерева!!!
    Осуществляется проход по строке. На данный момент находимся:   4
    Записываем символ в правый узел дерева!!!
   Осуществляется проход по строке. На данный момент находимся:   )
```

```
  Осуществляется проход по строке. На данный момент находимся:   )
 Осуществляется проход по строке. На данный момент находимся:   +
 Записываем символ в корневой узел дерева!!!
  Осуществляется проход по строке. На данный момент находимся:   (
   Осуществляется проход по строке. На данный момент находимся:   (
    Осуществляется проход по строке. На данный момент находимся:   5
    Записываем символ в левый узел дерева!!!
   Осуществляется проход по строке. На данный момент находимся:   +
   Записываем символ в корневой узел дерева!!!
    Осуществляется проход по строке. На данный момент находимся:   6
    Записываем символ в правый узел дерева!!!
   Осуществляется проход по строке. На данный момент находимся:   )
  Осуществляется проход по строке. На данный момент находимся:   -
  Записываем символ в корневой узел дерева!!!
   Осуществляется проход по строке. На данный момент находимся:   (
    Осуществляется проход по строке. На данный момент находимся:   7
    Записываем символ в левый узел дерева!!!
   Осуществляется проход по строке. На данный момент находимся:   *
   Записываем символ в корневой узел дерева!!!
    Осуществляется проход по строке. На данный момент находимся:   8
    Записываем символ в правый узел дерева!!!
   Осуществляется проход по строке. На данный момент находимся:   )
  Осуществляется проход по строке. На данный момент находимся:   )
 Осуществляется проход по строке. На данный момент находимся:   )
Функция makeTreeLKP завершает работу!
```

```
Дерево, в порядке КЛП:
+*-12+34-+56*78
```

```
Дерево:
          8
        *
          7
      -
          6
        +
          5
   +
          4
        +
          3
      *
          2
        -
          1
```

Тест6 – порядок ЛПК:

```
Выберите действие:
0 - выход
1 - ввод с файла в порядке ЛКП
2 - ввод с файла в порядке ЛПК
3 - ввод с консоли в порядке ЛКП
4 - ввод с консоли в порядке ЛПК
4
Пожалуйста, введите выражение.
((qwe)(rty)u)
Запускается функция makeTreeLPK!
 Осуществляется проход по строке. На данный момент находимся:   (
  Осуществляется проход по строке. На данный момент находимся:   (
   Осуществляется проход по строке. На данный момент находимся:   q
   Записываем символ в левый узел дерева!!!
   Осуществляется проход по строке. На данный момент находимся:   w
   Записываем символ в правый узел дерева!!!
  Осуществляется проход по строке. На данный момент находимся:   e
  Записываем символ в корневой узел дерева!!!
  Осуществляется проход по строке. На данный момент находимся:   )
  Осуществляется проход по строке. На данный момент находимся:   (
   Осуществляется проход по строке. На данный момент находимся:   r
   Записываем символ в левый узел дерева!!!
   Осуществляется проход по строке. На данный момент находимся:   t
   Записываем символ в правый узел дерева!!!
  Осуществляется проход по строке. На данный момент находимся:   y
  Записываем символ в корневой узел дерева!!!
  Осуществляется проход по строке. На данный момент находимся:   )
 Осуществляется проход по строке. На данный момент находимся:   u
 Записываем символ в корневой узел дерева!!!
 Осуществляется проход по строке. На данный момент находимся:   )
```

```
Дерево, в порядке КЛП:
ueqwyrt
```

```
Дерево:
         t
      y
         r
   u
         w
      e
         q
```

Тест 7(на крайние значения) – порядок ЛКП

```
Выберите действие:
0 - выход
1 - ввод с файла в порядке ЛКП
2 - ввод с файла в порядке ЛПК
3 - ввод с консоли в порядке ЛКП
4 - ввод с консоли в порядке ЛПК
3
Пожалуйста, введите выражение.
(abc)
Запускается функция makeTreeLKP!
 Осуществляется проход по строке. На данный момент находимся:   (
  Осуществляется проход по строке. На данный момент находимся:   a
  Записываем символ в левый узел дерева!!!
 Осуществляется проход по строке. На данный момент находимся:   b
 Записываем символ в корневой узел дерева!!!
  Осуществляется проход по строке. На данный момент находимся:   c
  Записываем символ в правый узел дерева!!!
 Осуществляется проход по строке. На данный момент находимся:   )
Функция makeTreeLKP завершает работу!
Дерево, в порядке ЛПК:
acb
Дерево, в порядке КЛП:
bac
Дерево, в порядке ЛКП:
abc
Дерево:
      c
   b
      a
```

**Вывод.**

В ходе работы была изучена структура данных бинарное дерево. Так же были получены навыки его создания, обхода и работы с ним.

## ПРИЛОЖЕНИЕ.

## КОД ПРОГРАММЫ.

# main.cpp

```
#include "Header.h"
#include <iostream>
#include<fstream>
#include<string.h>
#include <list>
#include <iomanip>      // std::setw
#include <fstream>
#include <cstdlib>

using namespace std;
using namespace binTree_modul;



void printLPK(node<char>* b);
bool isNull(node<char>* b);
char RootBT(node<char>* b); // для непустого бин.дерева
node<char>* Left(node<char>* b);// для непустого бин.дерева
node<char>* Right(node<char>* b);// для непустого бин.дерева


//____________________LKP_____________________________________________________
_____
node<char>* makeTreeLKP(string in, int& ptr, node<char>*&root, int& err, int& counter, int& t,
ofstream& fout)
{
	if (in[ptr] == '\0')
		return nullptr;
	root = new node<char>;//создаем узел
	cout << setw(counter + 1) << ' ' << "Осуществляется проход по строке. На данный
момент находимся:   " << in[ptr] << endl;
	fout << setw(counter + 1) << ' ' << "Осуществляется проход по строке. На данный
момент находимся:   " << in[ptr] << endl;
	if (in[ptr] == '(')
	{
		t = 1;
		counter++;
		ptr++;
		makeTreeLKP(in, ptr, root->lt, err, counter, t, fout);
		counter--;
		if (err || in[ptr] == ')')
		{
			cout << "Ошибка! после левого узла не введен корневой, либо его пустое
обозначение!" << endl;
			fout << "Ошибка! после левого узла не введен корневой, либо его пустое
обозначение!" << endl;
```

```
                err = 3;
                return root;
        }
        if (err)
                return root;
        if (in[ptr] == '#')// проверка
        {
                cout << "Ошибка! Пустая голова!" << endl;
                fout << "Ошибка! Пустая голова!" << endl;
                err = 2;
                return root;
        }

        cout << setw(counter + 1) << ' ' << "Осуществляется проход по строке. На
данный момент находимся:   " << in[ptr] << endl;
        cout << setw(counter + 1) << ' ' << "Записываем символ в корневой узел
дерева!!!" << endl;
        fout << setw(counter + 1) << ' ' << "Осуществляется проход по строке. На
данный момент находимся:   " << in[ptr] << endl;
        fout << setw(counter + 1) << ' ' << "Записываем символ в корневой узел
дерева!!!" << endl;
        root->info = in[ptr++];
        if (err || in[ptr] == ')')
        {
                cout << "Ошибка! после корневого узла не введен правый узел!" << endl;
                fout << "Ошибка! после корневого узла не введен правый узел!" << endl;
                err = 3;
                return root;
        }
        t = 2;
        counter++;
        makeTreeLKP(in, ptr, root->rt, err, counter, t, fout);
        counter--;
        if (err)
                return root;
        if (in[ptr] == ')')
        {
                cout << setw(counter + 1) << ' ' << "Осуществляется проход по строке. На
данный момент находимся:   " << in[ptr] << endl;
                fout << setw(counter + 1) << ' ' << "Осуществляется проход по строке. На
данный момент находимся:   " << in[ptr] << endl;
                if (in[ptr] != '\0')
                ptr++;

        }
        else
                err = 1;
        return root;
}
else
{
        if (t == 1)
```

```
		{
			cout << setw(counter + 1) << ' ' << "Записываем символ в левый узел дерева!!!" << endl;
			fout << setw(counter + 1) << ' ' << "Записываем символ в левый узел дерева!!!" << endl;
		}
		if (t == 2)
		{
			cout << setw(counter + 1) << ' ' << "Записываем символ в правый узел дерева!!!" << endl;
			fout << setw(counter + 1) << ' ' << "Записываем символ в правый узел дерева!!!" << endl;

		}
		root->info = in[ptr++];

		//return root;
	}
	return root;
}

//_____________________________________________LPK_____________________________

node<char>* makeTreeLPK(string in, int& ptr, node<char>*&root, int& err, int& counter, int& t, ofstream& fout)
{

	if (in[ptr] == '\0')
		return nullptr;
	root = new node<char>;//создаем узел
	cout<< setw(counter + 1) << ' ' << "Осуществляется проход по строке. На данный момент находимся:   " << in[ptr] << endl;
	fout << setw(counter + 1) << ' ' << "Осуществляется проход по строке. На данный момент находимся:   " << in[ptr] << endl;

	if (in[ptr] == '(')
	{
		t=1;
		ptr++;
		counter++;
		makeTreeLPK(in, ptr, root->lt, err, counter, t, fout);
		counter--;
		if (err || in[ptr] == ')')
		{
			cout << "Ошибка! после левого узла не введен правый, либо его пустое обозначение!" << endl;
			fout << "Ошибка! после левого узла не введен правый, либо его пустое обозначение!" << endl;
			err = 3;
			return root;
		}
		counter++;
```

```
		t = 2;
		makeTreeLPK(in, ptr, root->rt, err, counter, t, fout);
		counter--;

		if (err || in[ptr] == ')')
		{
			cout << "Ошибка! после правого узла не введен корень!" << endl;
			fout << "Ошибка! после правого узла не введен корень!" << endl;
				err = 3;
			return root;
		}
		if (err)
			return root;
		if (in[ptr] == '#')
		{
			cout << "Ошибка! Пустая голова!" << endl;
			fout << "Ошибка! Пустая голова!" << endl;
			err = 2;
				return root;
		}
		t=0;

		cout << setw(counter + 1) << ' ' << "Осуществляется проход по строке. На данный момент находимся:   " << in[ptr] << endl;
		cout << setw(counter + 1) << ' ' << "Записываем символ в корневой узел дерева!!!" << endl;
		fout << setw(counter + 1) << ' ' << "Осуществляется проход по строке. На данный момент находимся:   " << in[ptr] << endl;
		fout << setw(counter + 1) << ' ' << "Записываем символ в корневой узел дерева!!!" << endl;
		root->info = in[ptr++];
			if (in[ptr] == ')')
			{
				cout << setw(counter + 1) << ' ' << "Осуществляется проход по строке. На данный момент находимся:   " << in[ptr] << endl;
				fout << setw(counter + 1) << ' ' << "Осуществляется проход по строке. На данный момент находимся:   " << in[ptr] << endl;
				if(in[ptr] != '\0')
					ptr++;
			}
			else
				err = 1;
			return root;
	}

	else
	{

		if (t == 1)
		{
			cout << setw(counter + 1) << ' ' << "Записываем символ в левый узел дерева!!!" << endl;
```

```
                        fout << setw(counter + 1) << ' ' << "Записываем символ в левый узел
дерева!!!" << endl;
                }
                if (t == 2)
                {
                        cout << setw(counter + 1) << ' ' << "Записываем символ в правый узел
дерева!!!" << endl;
                        fout << setw(counter + 1) << ' ' << "Записываем символ в правый узел
дерева!!!" << endl;
                }
                        root->info = in[ptr++];

        }
        return root;
}

        bool isNull(node<char>* b)
        {
                return (b == NULL);
        }


        char RootBT(node<char>* b)                      // для непустого бин.дерева
        {
                if (b == NULL) { cerr << "Error: RootBT(null) \n"; exit(1); }
                else return b->info;
        }


        node<char>* Left(node<char>* b)         // для непустого бин.дерева
        {
                if (b == NULL) { cerr << "Error: Left(null) \n"; exit(1); }
                else return b->lt;
        }


        node<char>* Right(node<char>* b)        // для непустого бин.дерева
        {
                if (b == NULL) { cerr << "Error: Right(null) \n"; exit(1); }
                else return b->rt;
        }



        void printLPK(node<char>* b, ofstream& fout)
        {
                if (!isNull(b)) {

                        printLPK(Left(b), fout);
                        printLPK(Right(b), fout);
                        cout << RootBT(b);
                        fout << RootBT(b);
```

```
        }
    }

    void printKLP(node<char>* b, ofstream& fout)
    {
        if (!isNull(b)) {
            cout << RootBT(b);
            fout << RootBT(b);
            printKLP(Left(b), fout);
            printKLP(Right(b), fout);
        }
    }

    void printLKP(node<char>* b, ofstream& fout)
    {
        if (!isNull(b)) {
            printLKP(Left(b), fout);
            cout << RootBT(b);
            fout << RootBT(b);
            printLKP(Right(b), fout);
        }
    }

    void printTree(node<char>* p, int level, ofstream& fout)//функция, для изображения
дерева на экран
    {
        if (p)
        {
            printTree(p->rt, level + 1, fout);
            for (int i = 0; i < level; i++)
            {
                cout << "   ";
                fout << "   ";
            }
            cout << p->info << endl;
            fout << p->info << endl;
            printTree(p->lt, level + 1, fout);
        }
    }

    void destroy(node<char>*& b)
    {
        if (b != NULL) {
            destroy(b->lt);
            destroy(b->rt);
            delete b;
            b = NULL;
        }
    }
```

```
int main()
{
	setlocale(LC_ALL, "rus");
	int f = 0;
	int choice;//переменная, для выбора действий
	string str;
	string out;
	string two;
	int i = 0;
	int k = 1;
	int t = 0;
	int counter = 0;
	int err = 0;
	ofstream fout("endfile.txt"); // создаём объект класса ofstream для записи и связываем его с файлом endfile.txt
	cout << "Выберите действие:" << endl << "0 - выход" << endl << "1 - ввод с файла в порядке ЛКП" << endl << "2 - ввод с файла в порядке ЛПК" << endl << "3 - ввод с консоли в порядке ЛКП" << endl << "4 - ввод с консоли в порядке ЛПК" << endl;
	cin >> choice;
	cin.ignore();
	if (choice == 1 || choice == 2 || choice == 0 || choice == 3 || choice == 4)
	{
		if (choice == 0)
		{
			cout << "До свидания!" << endl;
			fout << "До свидания!" << endl;
			return 0;
		}

		else if (choice == 1 || choice == 2)
		{
			if (choice == 1)
				f = 1;//флаг, для обозначения порядка ЛКП
			else f = 2;//ЛПК
			string name;

			cout << "Пожалуйста, введите название файла." << endl;
			cin >> name;
			cin.ignore();
			char buff[50]; // буфер промежуточного хранения считываемого из файла текста
			ifstream fin("D:/вижак/Project7/Project7/test.txt"); // открыли файл для чтения

			if (!fin)
			{
				cout << "Ошибка открытия файла." << endl;
				fout << "Ошибка открытия файла." << endl;
				return 1;
			}

			else
```

```
		{
			cout << "Всё отлично, файл открылся!" << endl;
			fout << "Всё отлично, файл открылся!" << endl;
		}

		fin.getline(buff, 50); // считали строку из файла
		fin.close(); // закрываем файл
		str.append(buff);
		cout << str << endl;
	}



	else if (choice == 3 || choice == 4)
	{
		if (choice == 3)
			f = 1;//флаг, для обозначения порядка ЛКП
		else f = 2;//ЛПК
		cout << "Пожалуйста, введите выражение." << endl;
		//fout<< "Пожалуйста, введите выражение." << endl;
		char s[100];
		cin.getline(s, 100);
		str.append(s);
	}
}

else
{
	cout << "Неверный ввод" << endl;

}

node<char>* tree = NULL;
//node<char>* tr = NULL;
if (f == 1)//если флаг 1, то вызываем функцию для обхода ЛКП
{
	cout << "Запускается функция makeTreeLKP! " << endl;
	fout << "Запускается функция makeTreeLKP! " << endl;
	tree = makeTreeLKP(str, i, tree, err, counter, t, fout);
	cout << "Функция makeTreeLKP завершает работу! " << endl;
	fout << "Функция makeTreeLKP завершает работу! " << endl;
}
else//иначе, флаг равен 2, т е вызывается функция для обхода ЛПК
{
	cout << "Запускается функция makeTreeLPK! " << endl;
	fout << "Запускается функция makeTreeLPK! " << endl;
	tree = makeTreeLPK(str, i, tree, err, counter, t, fout);
	cout << "Функция makeTreeLPK завершает работу! " << endl;
	fout << "Функция makeTreeLPK завершает работу! " << endl;
}

if (err == 0)
```

```
	{
		cout << "Дерево, в порядке ЛПК:" << endl;
		fout << "Дерево, в порядке ЛПК:" << endl;
		printLPK(tree, fout);
		cout << endl;
		cout << "Дерево, в порядке КЛП:" << endl;
		fout << endl;
		fout << "Дерево, в порядке КЛП:" << endl;
		printKLP(tree, fout);
		cout << endl;
		cout << "Дерево, в порядке ЛКП:" << endl;
		fout << endl;
		fout << "Дерево, в порядке ЛКП:" << endl;
		printLKP(tree, fout);
		cout << endl;
		cout << "Дерево:" << endl;
		fout << endl;
		fout << "Дерево:" << endl;
		printTree(tree, k, fout);
	}

	fout.close();
	return 0;

}
```

## Header.h

```
#include <cstddef>
#include<string.h>
#include <string>
using namespace std;
namespace binTree_modul
{
	template <typename T>



	struct node {//структура бинарного дерева
		T info;//информация что хранится в узле бд
		node* lt;//указатель на левый корень
		node* rt;//указатель на правый корень
		// constructor
		node() { lt = NULL; rt = NULL; }
	};
```

```
} // end of namespace binTree_modul
#pragma once
```