

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

**ОТЧЕТ**  
**по лабораторной работе №5**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Бинарные деревья поиска, AVL деревья**

Студентка гр. 8383

Сырцова Е.А.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2019

## **Цель работы**

Познакомиться со структурой и реализацией бинарного дерева поиска на языке программирования C++.

## **Формулировка задачи**

Вариант 18. БДП: AVL дерево – действия 1+2в.

- 1) По заданному файлу F (типа file of Elem), все элементы которого различны, построить структуру типа БДП;
- 2) Записать в файл элементы построенного БДП в порядке их возрастания; вывести построенное БДП на экран.

## **Реализация задачи**

В данной работе используются структуры elem и trunk:

Описание структуры elem – структура узлов дерева, где

- int val – значение узла;
- int height – уровень узла в дереве;
- elem\* left – узел левого поддерева;
- elem\* right – узел правого поддерева.

Описание структуры trunk – структура связей дерева, где

- trunk\* prev – связь с предыдущими узлами;
- string str – строка, отображающая связь узлов в дереве.

## **Описание алгоритма и функций**

В функции main() вызывается функция menu(), которая выводит на экран меню выбора ввода данных, в случае ввода единицы программа вызывает функцию работы с консолью console(), в случае ввода двойки программа вызывает функцию работы с файлом file(), в случае ввода нуля программа завершает работу и в случае другого ввода – генерирует ошибку.

Работа с консолью: вводится строка элементов дерева, проверяется на корректность функцией checkStr.

- `int checkStr(string str)` – все элементы строки `str` – натуральные числа, возвращает 1 при корректном вводе, 0 при некорректном вводе.

Для ввода с консоли дополнительно проверяются отрицания (знак минуса не является концом строки и после него обязательно есть число). При корректном вводе создается БДП, где первый элемент – первый узел. Пока не встретится конец строки вызываются функции:

- `void addToTree(elem* root, int tmp_val, int count)`, где
  - `root` – корень дерева;
  - `tmp_val` – значение нового узла;
  - `count` – счётчик глубины рекурсии.

Функция сравнивает значение нового узла со значением корня, если новый узел больше корня – вызывает сама себя для правого поддерева, если оно не пустое, если правое поддерево пустое – создает правое поддерево с новым узлом и пустыми поддеревьями. Если новый узел меньше корня – вызывает сама себя для левого поддерева, если оно не пустое, если левое поддерево пустое – создает левое поддерево с новым узлом и пустыми поддеревьями. Выводятся сообщения о сравнении значений узлов с рекурсивными отступами.

- `void printTree(elem* tree, trunk* prev, bool isRight)`, где
  - `tree` – корень дерева;
  - `prev` – связь с предыдущими узлами;
  - `isRight` – метка правых поддеревьев для выбора связей в дереве.

Рекурсивная функция, вызывает сама себя для правого поддерева, меняя метку правого поддерева, выводит узел и вызывает сама себя для вывода левого поддерева, обнуляя метку правого поддерева.

Функция выводит дерево с соответствующими связями (`-->` корень дерева, `.->` правое поддерево, ``-->` левое поддерево, добавляя `|` для широких деревьев), вызывая для этого рекурсивную функцию `showTrunk`.

- `void showTrunk(trunk* p, int &count)`, где

- `p` – текущая строка связи;
- `count` – счётчик уровней узлов.

Функция вызывает сама себя, увеличивая счетчик уровня, выводит связи между узлами дерева.

- `int checkTree(elem* root)`, где
- `root` – корень дерева.

Функция проверяет баланс дерева, высота левых и правых поддеревьев в АВЛ дереве различаются не более чем на единицу. Т.е. функция рекурсивно вычисляет высоту левого и правого поддерева, возвращает нуль, если дерево сбалансированно, или значение узла, в котором нарушено условие АВЛ дерева.

Если дерево не сбалансированно вызывается функция `fixTree`.

- `elem* fixTree(elem* root, int tmp)`, где
- `root` – корень дерева;
- `tmp` – узел с нарушенным балансом.

Функция доходит до узла с нарушенным балансом, если высота правого поддерева больше – выполняет левый поворот и снова проверяет дерево, если дерево не сбалансировано – выводит промежуточный результат и выполняет большой правый поворот. Если высота левого поддерева больше – выполняет правый поворот и снова проверяет дерево, если дерево не сбалансировано – выводит промежуточный результат и выполняет большой левый поворот.

Выводится сбалансированное дерево. В результате построено АВЛ дерево, на консоль выводится БДП, элементы дерева в порядке возрастания записываются в файл `output.txt` функцией `outElem`.

- `void outElem(ofstream& f2, elem* root)`, где
- `f2` – поток вывода;
- `root` – корень дерева.

Функция сначала вызывает сама себя для левого поддерева, затем выводит корень дерева и вызывает себя для правого поддерева.

Удаляется построенное БДП.

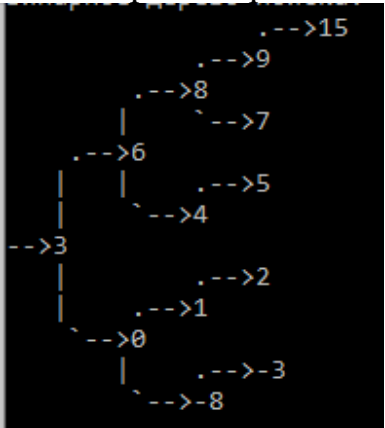
- void delTree(elem\* root), где
  - root – корень дерева.

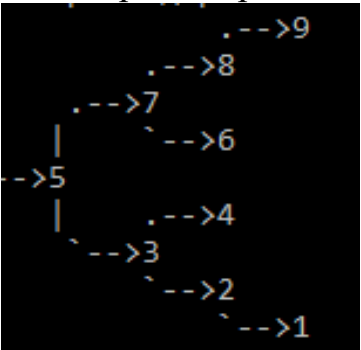
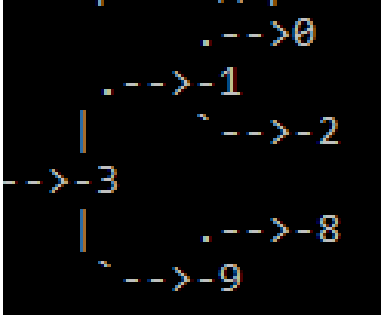
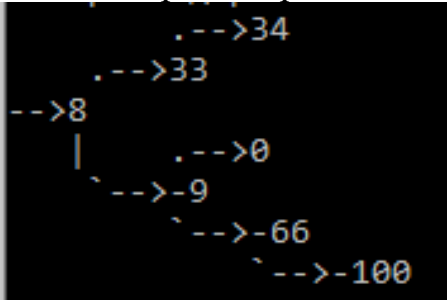
Функция рекурсивно удаляет левое и правое поддеревья.

В основной функции удаляется корень дерева.

Работа с файлом использует такой же алгоритм, за исключением считывания. Вводится строка с названием файла ввода, элементы дерева считываются из этого файла, если файл не открылся или файл пуст, функция генерирует ошибку.

### Тестирование

Входные данные	Выходные данные
<pre>2 input.txt 1 9 2 8 3 7 6 4 5 -8 0 15 -3</pre>	<p>Бинарное дерево:</p>  <p>Элементы построенного БДП в порядке их возрастания: -8 -3 0 1 2 3 4 5 6 7 8 9 15</p>
<pre>2 wow.docx 8 -5 0 - 7</pre>	неверное выражение
<pre>2 wow.docx</pre>	неверное выражение
<pre>1 5 0 -8 9 -</pre>	неверное выражение
<pre>1 2 5 *</pre>	неверное выражение
<pre>1 a b c</pre>	неверное выражение, элементами дерева могут быть только натуральные числа

3	Неверно введены данные!
1	неверное выражение
<div>1</div> <div>1 2 3 4 5 6 7 8 9</div>	<div>Бинарное дерево:</div>  <div>Элементы построенного БДП в порядке их возрастания: 1 2 3 4 5 6 7 8 9</div>
<div>1</div> <div>-2 -9 -3 0 -8 -1</div>	<div>Бинарное дерево:</div>  <div>Элементы построенного БДП в порядке их возрастания: -9 -8 -3 -2 -1 0</div>
<div>1</div> <div>34 33 -9 8 -66 0 -100</div>	<div>Бинарное дерево:</div>  <div>Элементы построенного БДП в порядке их возрастания:</div>

## Пример промежуточных выводов

```
C:\Users\user\Desktop\5\Debug\5.exe
1 9 2 8 3 7 6 4 5 -8 0 15 -3
Первый элемент: 1
Добавляем следующий элемент 9 в дерево
Новый элемент больше узла: 1 < 9. Добавляем элемент в правое поддерево.
--->9
-->1
Проверяем дерево на авл
Дерево сбалансированно
Добавляем следующий элемент 2 в дерево
Новый элемент больше узла: 1 < 2. Добавляем элемент в правое поддерево.
Новый элемент меньше узла: 9 > 2. Добавляем элемент в левое поддерево.
    --->9
    |    ^-->2
-->1
Проверяем дерево на авл
Сбалансируем дерево
левый поворот узла 1
--->9
|    ^-->2
|    ^-->1
большой правый поворот узла 9
    --->9
    ^-->2
    ^-->1
Дерево сбалансированно
Добавляем следующий элемент 8 в дерево
Новый элемент больше узла: 2 < 8. Добавляем элемент в правое поддерево.
Новый элемент меньше узла: 9 > 8. Добавляем элемент в левое поддерево.
    --->9
    |    ^-->8
-->2    ^-->1
Проверяем дерево на авл
Дерево сбалансированно
Добавляем следующий элемент 3 в дерево
Новый элемент больше узла: 2 < 3. Добавляем элемент в правое поддерево.
Новый элемент меньше узла: 9 > 3. Добавляем элемент в левое поддерево.
Новый элемент меньше узла: 8 > 3. Добавляем элемент в левое поддерево.
    --->9
    |    ^-->8
    |    ^-->3
-->2    ^-->1
```

```
C:\Users\user\Desktop\5\Debug\5.exe
    ^-->1
Проверяем дерево на авл
Сбалансируем дерево
левый поворот узла 2
--->9
|    ^-->8
|    |    ^-->3
|    ^-->2
|    ^-->1
большой правый поворот узла 9
    --->9
    ^-->8
    |    ^-->3
    |    ^-->2
    |    ^-->1
Дерево сбалансированно
Добавляем следующий элемент 7 в дерево
Новый элемент меньше узла: 8 > 7. Добавляем элемент в левое поддерево.
Новый элемент больше узла: 2 < 7. Добавляем элемент в правое поддерево.
Новый элемент больше узла: 3 < 7. Добавляем элемент в правое поддерево.
    --->9
    ^-->8
    |    ^-->7
    |    ^-->3
    |    ^-->2
    |    ^-->1
Проверяем дерево на авл
Сбалансируем дерево
правый поворот узла 8
    --->9
    ^-->8
    |    ^-->7
    |    ^-->3
-->2    ^-->1
большой левый поворот узла 2
    --->9
    ^-->8
    |    ^-->7
-->3    ^-->2
    ^-->1
Дерево сбалансированно
Добавляем следующий элемент 6 в дерево
```

```

Дерево сбалансированно
Добавляем следующий элемент 6 в дерево
Новый элемент больше узла: 3 < 6. Добавляем элемент в правое поддерево.
    Новый элемент меньше узла: 8 > 6. Добавляем элемент в левое поддерево.
        Новый элемент меньше узла: 7 > 6. Добавляем элемент в левое поддерево.
            .-->9
            |
            .-->8
            |
            .-->7
            |
            .-->6
            |
        -->3
        |
        .-->2
        |
        .-->1
Проверяем дерево на авл
Дерево сбалансированно
Добавляем следующий элемент 4 в дерево
Новый элемент больше узла: 3 < 4. Добавляем элемент в правое поддерево.
    Новый элемент меньше узла: 8 > 4. Добавляем элемент в левое поддерево.
        Новый элемент меньше узла: 7 > 4. Добавляем элемент в левое поддерево.
            Новый элемент меньше узла: 6 > 4. Добавляем элемент в левое поддерево.
                .-->9
                |
                .-->8
                |
                .-->7
                |
                .-->6
                |
            -->3
            |
            .-->2
            |
            .-->1
Проверяем дерево на авл
Сбалансируем дерево
левый поворот узла 3
    .-->9
    |
    .-->8
    |
    .-->7
    |
    .-->6
    |
    .-->4
    |
    .-->3
    |
    .-->2
    |
    .-->1
большой правый поворот узла 8

```

```

    .-->9
    .-->8
-->7
|
| .-->6
| |
| | .-->4
| | |
| | | .-->3
| | | |
| | | | .-->2
| | | | |
| | | | | .-->1
Дерево сбалансированно
Добавляем следующий элемент 5 в дерево
Новый элемент меньше узла: 7 > 5. Добавляем элемент в левое поддерево.
    Новый элемент больше узла: 3 < 5. Добавляем элемент в правое поддерево.
        Новый элемент меньше узла: 6 > 5. Добавляем элемент в левое поддерево.
            Новый элемент больше узла: 4 < 5. Добавляем элемент в правое поддерево.
                .-->9
                .-->8
-->7
|
| .-->6
| |
| | .-->5
| | |
| | | .-->4
| | | |
| | | | .-->3
| | | | |
| | | | | .-->2
| | | | | |
| | | | | | .-->1
Проверяем дерево на авл
Сбалансируем дерево
правый поворот узла 7
    .-->9
    .-->8
    .-->7
    |
    | .-->6
    | |
    | | .-->5
    | | |
    | | | .-->4
-->3
|
| .-->2
| |
| | .-->1
большой левый поворот узла 3
    .-->9
    .-->8
    .-->7
-->6
|
| .-->5
| |
| | .-->4
| | |
| | | .-->3
| | | |
| | | | .-->2

```



```

сбалансируем дерево
левый поворот узла 7
    .-->9
    |
    .-->8
    |
    .-->6
    |
    .-->5
    |
    .-->4
    |
    .-->3
    |
    .-->2
    |
    .-->1
Дерево сбалансированно
Добавляем следующий элемент -8 в дерево
Новый элемент меньше узла: 6 > -8. Добавляем элемент в левое поддерево.
    Новый элемент меньше узла: 3 > -8. Добавляем элемент в левое поддерево.
        Новый элемент меньше узла: 2 > -8. Добавляем элемент в левое поддерево.
            Новый элемент меньше узла: 1 > -8. Добавляем элемент в левое поддерево.
                .-->9
                |
                .-->8
                |
                .-->6
                |
                .-->5
                |
                .-->4
                |
                .-->3
                |
                .-->2
                |
                .-->1
                |
                .-->-8
Проверяем дерево на авл
Сбалансируем дерево
правый поворот узла 6
    .-->9
    |
    .-->8
    |
    .-->7
    |
    .-->6
    |
    .-->5
    |
    .-->4
    |
    .-->3
    |
    .-->2
    |
    .-->1
    |
    .-->-8
Сбалансируем дерево
правый поворот узла 2

```

```

      .-->9
        |
    .--->8
       | \
     --->7
         |
   .---->6
          | \
        -->5
           | \
        ---->4
            |
    -.->3
      | 
    .--->2
       | \
     -->1
        | \
      -->-8

```

Дерево сбалансированно  
Добавляем следующий элемент 0 в дерево  
Новый элемент меньше узла: 3 > 0. Добавляем элемент в левое поддерево.  
    Новый элемент меньше узла: 1 > 0. Добавляем элемент в левое поддерево.  
        Новый элемент больше узла: -8 < 0. Добавляем элемент в правое поддерево.

```

      .-->9
        |
    .--->8
       | \
     --->7
         |
   .---->6
          | \
        -->5
           | \
        ---->4
            |
    -.->3
      | 
    .--->2
       | \
     -->1
        | \
      -->0
        | \
      -->-8

```

Проверяем дерево на авл  
Дерево сбалансированно  
Добавляем следующий элемент 15 в дерево  
Новый элемент больше узла: 3 < 15. Добавляем элемент в правое поддерево.  
    Новый элемент больше узла: 6 < 15. Добавляем элемент в правое поддерево.  
        Новый элемент больше узла: 8 < 15. Добавляем элемент в правое поддерево.  
            Новый элемент больше узла: 9 < 15. Добавляем элемент в правое поддерево.

```

      .-->15
        |
    .--->9
        |
    .--->8
       | \
     --->7
         |
   .---->6
          | \
        -->5
           | \
        ---->4
            |
    -.->3
      | 
    .--->2
       | \
     -->1
        | \
      -->0

```



## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include "pch.h"
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <cstring>
#include <cmath>
#include <cstdlib>
#include <cctype>
using namespace std;

void menu() { //выбор опций
    cout << "Выберите действие:" << endl;
    cout << "1 - ввод с клавиатуры" << endl;
    cout << "2 - ввод из файла" << endl;
    cout << "0 - выход" << endl;
}

struct elem { //структура узла авл дерева
    int val; //значение узла
    int height; //высота
    elem* left; //левый узел
    elem* right; //правый узел
};

struct trunk { //структура - ствол дерева
    trunk* prev; //предыдущий узел - корень
    string str; //строка отступов (уровень узла)
    trunk(trunk* prev, string str) {
        this->prev = prev;
        this->str = str;
    }
};

void showTrunk(trunk* p, int &count) { //функция вывода связей
    дерева
    if (p == NULL) //если нет поддеревьев
        return;
    showTrunk(p->prev, count); //выводим предыдущий узел
    count++; //увеличиваем уровень
    cout << p->str ; //выводим отступы и связи
}
```

```

void printTree(elem* tree, trunk* prev, bool isRight) { //функция
выводит узлы дерева, вызывает функцию вывода связей
    if (tree == NULL) //пустое дерево
        return;
    string prev_str = "    "; //отступ по уровням (длина как для
стрелки)
    trunk* tmp = new trunk(prev, prev_str); //новая связь
    printTree(tree->right, tmp, 1); //правое поддерево
    if (!prev) //если нет предыдущего узла (предка) -> корень
дерева
        tmp->str = "-->"; //связь корня дерева
    else if (isRight) { //если правое поддерево
        tmp->str = ".-->"; //связь правого поддерева
        prev_str = "    |"; //в отступ по уровням добавляем
черту (дерево идет вширь)
    }
    else { //в противном случае - левое поддерево
        tmp->str = "`-->"; //левое поддерево
        prev->str = prev_str; //отступ по уровням не меняется
    }
    int count = 0; //уровень узла
    showTrunk(tmp, count); //выводим связи дерева - стебли
    cout << tree->val << endl; //выводим значение узла
    if (prev) //задаем строку отступов для узла, если есть
поддерева
        prev->str = prev_str;
    tmp->str = "    |"; //в отступ по уровням добавляем черту
(дерево идет вширь)
    printTree(tree->left, tmp, 0); //левое поддерево
}

int height(elem* root) { //находит высоту дерева с корнем root
    int a = 0; //высота левого поддерева
    int b = 0; //высота правого поддерева
    if (root->left != NULL)
        a = height(root->left); //находим высоту левого
поддерева
    if (root->right != NULL)
        b = height(root->right); //находим высоту правого
поддерева
    if (a < b)
        a = b; //высота поддерева для корня - высота
наибольшего поддерева
}

```

```

        root->height = a + 1;//высота дерева - высота наибольшего
поддерева +1
        return root->height;//возвращает высоту дерева
    }

int checkTree(elem* root) {//проверка дерева на авл
    int tmp = 0;//метка - является ли авл деревом, 0-да
    int left = 0;//высота левого поддерева
    int right = 0;//высота правого поддерева
    if (root->left != NULL)//если есть левое поддерево
        left = root->left->height;//высота левого поддерева
    if (root->right != NULL)//если есть правое поддерево
        right = root->right->height;//высота правого поддерева
    if (abs(left - right) > 1)//если разница высот левого и
правого поддерева больше 1 - не авл дерево
        return root->val;//возвращаем значение корня -> не авл
дерево
    else {
        if (root->left != NULL)//если есть левое поддерево
            tmp += checkTree(root->left);//проверяем левое
поддерево на авл
        if (root->right != NULL)//если есть правое поддерево
            tmp += checkTree(root->right);//проверяем правое
поддерево на авл
    }
    return tmp;//возвращаем метку
}

void leftTurn(elem* tmp, elem* root) {//малый левый поворот
(бывший "корень", новый "корень")
    tmp->right = root->left;//правое поддерево "корня" - левое
поддерево левого поддерева
    root->left = tmp;//левое поддерево нового "корня" - бывший
"корень"
}

void rightTurn(elem* tmp, elem* root) {//малый правый поворот
(бывший "корень", новый "корень")
    tmp->left = root->right;//левое поддерево "корня" - правое
поддерево правого поддерева
    root->right = tmp;//правое поддерево нового "корня" - бывший
"корень"
}

```

```

void bigRightTurn(elem* root, elem* tmp) { //большой правый
поворот (бывший "корень", новый "корень")
    tmp->left->right = root->left; //правое поддерево левого
поддерева левого поддерева - левое поддерево бывшего "корня"
    root->left = tmp->left; //левое поддерево бывшего "корня" -
левое поддерево нового корня
    tmp->left = root->right; //левое поддерево нового корня -
правое поддерево бывшего корня
    root->right = tmp; //правое поддерево бывшего корня - новый
корень
}

void bigLeftTurn(elem* root, elem* tmp) { //большой левый поворот
(бывший "корень", новый "корень")
    tmp->right->left = root->right; //левое поддерево правого
поддерева нового корня - правое поддерево бывшего корня
    root->right = tmp->right; //правое поддерево бывшего корня -
правое поддерево нового корня
    tmp->right = root->left; //правое поддерево нового корня -
левое поддерево корня
    root->left = tmp; //левое поддерево бывшего корня - новый
корень
}

elem* fixTree(elem* root, int tmp) {
    int left = 0; //высота левого поддерева
    int right = 0; //высота правого поддерева
    if (root->left != NULL) //если есть левое поддерево
        left = root->left->height; //его высота
    if (root->right != NULL) //если есть правое поддерево
        right = root->right->height; //его высота

    if (root->val == tmp) { //узел где была ошибка
        elem* tmp_elem = new elem;
        tmp_elem = root; //сохраняем узел
        if (right > left) { //высота правого поддерева больше
            root = tmp_elem->right; //корень-узел правого
поддерева

            cout << "левый поворот узла " << tmp << endl;
            leftTurn(tmp_elem, root); //левый поворот
            height(root); //новая высота
            tmp = checkTree(root); //проверяем новое дерево
            if (tmp == root->val) {
                printTree(root, NULL, 0); //после поворота
дерево не сбалансировалось
            }
        }
    }
}

```

```

        tmp_elem = root;
        root = tmp_elem->left->right;
        cout << "большой правый поворот узла " <<
tmp << endl;
        bigRightTurn(root, tmp_elem);
    }
}
else {
    root = tmp_elem->left; //корень-узел левого
поддерева
    cout << "правый поворот узла " << tmp << endl;
    rightTurn(tmp_elem, root); //правый поворот
    height(root); //новая высота
    tmp = checkTree(root); //проверка нового дерева
    if (tmp == root->val) {
        printTree(root, NULL, 0); //после поворота
дерево не сбалансировалось
        tmp_elem = root;
        root = tmp_elem->right->left;
        cout << "большой левый поворот узла " << tmp
<< endl;
        bigLeftTurn(root, tmp_elem);
    }
}
}
else {
    if (tmp > root->val) //если с корнем все в порядке
        root->right = fixTree(root->right,
tmp); //проверяем правое
    else
        root->left = fixTree(root->left, tmp); //и левое
поддерева
    }
    return root; //возвращает корень нового дерева
}

void addToTree(elem* root, int tmp_val, int count) { //добавление
в дерево следующего элемента
    if (root->val < tmp_val) { //если больше узла
        for (int i = count; i > 0; i--) cout << "\t"; //отступы
рекурсии
        cout << "Новый элемент больше узла: " << root->val <<
" < " << tmp_val << ". Добавляем элемент в правое поддерево." <<
endl;
        if (root->right != NULL) //и есть правое поддерево

```

```

        addToTree(root->right, tmp_val,
count+1);//добавляем в правое поддерево
        else {
            root->right = new elem;//если нет - создаем
правое поддерево
            root->right->left = NULL;
            root->right->right = NULL;
            root->right->val = tmp_val;//и добавляем узел
        }
    }
    else if (root->val > tmp_val) {//если меньше узла
        for (int i = count; i > 0; i--) cout << "\t";//отступы
рекурсии
        cout << "Новый элемент меньше узла: " << root->val <<
" > " << tmp_val << ". Добавляем элемент в левое поддерево." <<
endl;
        if (root->left != NULL)//есть левое поддерево
            addToTree(root->left, tmp_val,
count+1);//добавляем в левое поддерево
        else {
            root->left = new elem;//если нет - создаем левое
поддерево
            root->left->left = NULL;
            root->left->right = NULL;
            root->left->val = tmp_val;//и добавляем узел
        }
    }
}

void delTree(elem* root) {//удаляет дерево с корнем root
    if (root->left != NULL) {//удаляет левое поддерево
        delTree(root->left);
        delete(root->left);
    }
    if (root->right != NULL) {//удаляет правое поддерево
        delTree(root->right);
        delete(root->right);
    }
}

int checkStr(string str) {//проверка ввода
    for (int i = 0; i < str.length(); i++) {
        if (!isdigit(str[i]))//если есть не цифры в строке
возвращаем 0
    }
}

```



```

        if ((str[i] != '-') && (str[i] != ' '))//но это
отрицательные числа и разделители
        return 0;
    }
}
return 1;
}

void outElem(ofstream& f2, elem* root) {//вывод отсортированных
элементов в строку (в файле)
    if (root->left != NULL)//сначала левые поддеревья
        outElem(f2, root->left);
    f2 << root->val << " ";//узел и отступ
    if (root->right != NULL)//правые поддеревья
        outElem(f2, root->right);
}

int console() {//работа с консолью
    cout << "Введите элементы дерева через пробел" << endl;
    int tmp = 0;
    char str[256];
    string str1;//строка ввода
    elem* root = new elem;//первый элемент
    root->left = NULL;//без поддеревьев
    root->right = NULL;
    getline(cin, str1);//считываем строку ввода
    getline(cin, str1);
    if (checkStr(str1) == 0) {//проверка корректности
        cout << "неверное выражение, элементами дерева могут
быть только натуральные числа" << endl;
        return 0;
    }
    for (int i = 0; i < str1.length(); i++)//проверка на
отрицательные числа
        if (str1[i] == '-') {
            if (i == str1.length() - 1) {//если минус
последний в строке
                cout << "неверное выражение" << endl;
                return 0;
            }
            if (!isdigit(str1[i + 1])) {//если после минуса
нет числа
                cout << "неверное выражение" << endl;
                return 0;
            }
        }
}

```

```

    }
    int flag = 0; //флаг
    int current_s = 0; //текущий символ
    int current_c = 0;
    while (!flag) {
        if (str1[current_s] == ' ')
            current_s++;
        else
            flag = 1; //нет пробела (в числе несколько
СИМВОЛОВ)
        if (current_s == str1.length()) { //конец строки
            cout << "неверное выражение" << endl;
            return 0;
        }
    }
    while (flag) {
        if (isdigit(str1[current_s]) || (str1[current_s] == '-'
'')) {
            str[current_c] = str1[current_s];
            current_s++;
            current_c++;
        }
        else {
            flag = 0;
        }
    }
    tmp = atoi(str);
    for (int i = 0; i < current_c; i++)
        str[i] = 0;
    current_c = 0;
    cout << "Первый элемент: " << tmp << endl;
    root->val = tmp;
    height(root);
    int end_flag = 0;
    while (!end_flag) {
        while (!flag) {
            if (current_s == str1.length()) {
                end_flag = 1;
                break;
            }
            if (str1[current_s] == ' ')
                current_s++;
            else
                flag = 1;
        }
    }

```

```

    }
    if (end_flag)
        break;
    while (flag) {
        if (current_s == str1.length()) {
            end_flag = 1;
            break;
        }
        if (isdigit(str1[current_s]) || (str1[current_s]
== '-'')) {
            str[current_c] = str1[current_s];
            current_s++;
            current_c++;
        }
        else {
            flag = 0;
        }
    }
    tmp = atoi(str);
    for (int i = 0; i < current_c; i++)
        str[i] = 0;
    current_c = 0;
    cout << "Добавляем следующий элемент " << tmp << " в
дерево" << endl;
    addToTree(root, tmp, 0);
    printTree(root, NULL, 0);
    height(root); //новая высота
    cout << "Проверяем дерево на авл" << endl;
    tmp = checkTree(root);
    while (tmp) {
        cout << "Сбалансируем дерево" << endl;
        root = fixTree(root, tmp);
        printTree(root, NULL, 0);
        height(root);
        tmp = checkTree(root);
    }
    if (!tmp) cout << "Дерево сбалансированно" << endl;
}
cout << endl << "Бинарное дерево:" << endl;
printTree(root, NULL, 0);
ofstream f2;
f2.open("output.txt"); //открываем файл вывода
f2 << "Элементы построенного БДП в порядке их возрастания:"
<< endl;
outElem(f2, root); //выводим элементы

```

```

        f2.close();//закрываем файл
        delTree(root);//удаляем поддеревья
        delete(root);//удаляем корень
        cout << endl << "Элементы построенного БДП в порядке их
возрастания записаны в файле output.txt" << endl;
        return 0;
    }

    int file() { //работа с файлом
        cout << "Введите имя файла, в котором записаны элементы
дерева" << endl;
        string file_name;//имя файла
        cin >> file_name;
        ifstream f;//поток ввода
        f.open(file_name.c_str());
        if (!f) { //файл не открыт
            cout << "Файл не открыт!" << endl;
            return 0;
        }
        int tmp = 0;
        char str[256]; //строка в файле
        elem* root = new elem; //первый элемент
        f >> str;
        if (checkStr(str)) //корректность ввода
            tmp = atoi(str);
        else {
            cout << "неверное выражение" << endl;
            return 0;
        }
        cout << "Первый элемент: " << tmp << endl;
        root->val = tmp; //первый элемент
        root->left = NULL; //без поддеревьев
        root->right = NULL; //без поддеревьев
        height(root); //высота дерева
        while (!f.eof()) { //считываем строку до конца
            f >> str;
            if (checkStr(str)) //корректность ввода
                tmp = atoi(str);
            else {
                cout << "неверное выражение" << endl; //неверное
выражение
                return 0;
            }
            cout << "Добавляем следующий элемент " << tmp << " в
дерево" << endl;

```

```

        addToTree(root, tmp, 0);
        printTree(root, NULL, 0);
        height(root); //новая высота
        cout << "Проверяем дерево на авл" << endl;
        tmp = checkTree(root); //проверяем дерево
        while (tmp) {
            cout << "Сбалансируем дерево" << endl;
            root = fixTree(root, tmp);
            printTree(root, NULL, 0);
            height(root); //новая высота
            tmp = checkTree(root); //проверяем дерево опять
        }
        if (!tmp) cout << "Дерево сбалансированно" << endl;
    }
    cout << endl << "Бинарное дерево поиска:" << endl;
    printTree(root, NULL, 0);
    ofstream f2; //поток вывода
    f2.open("output.txt"); //файл вывода
    f2 << "Элементы построенного БДП в порядке их возрастания:"
    << endl;
    outElem(f2, root); //функция вывода в файл
    f2.close(); //закрываем файл вывода
    f.close(); //закрываем файл ввода
    delTree(root); //удаляем поддеревья
    delete(root); //удаляем корень
    cout << endl << "Элементы построенного БДП в порядке их
    возрастания записаны в файле output.txt" << endl;
    return 0;
}

int main() {
    system("chcp 1251"); //русский язык
    system("cls"); //очистить консоль
    menu(); //выводим меню
    char way; //выбор опции
    cin >> way;
    while (way != '0') { //0-выход из программы
        switch (way) {
            case '1':
                console(); //работа с консолью
                cout << endl;
                menu(); //снова меню
                cin >> way; //и выбор
                break;
            case '2':

```

```
        file();//работа с файлом
        cout << endl;
        menu();//снова меню
        cin >> way;//и выбор
        break;
    default:
        cout << "Неверно введены данные!" <<
endl;//неверный ввод
        menu();//меню
        cin >> way;//выбор
    }
}
return 0;
}
```