

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

**КУРСОВАЯ РАБОТА**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: AVL-деревья**

Студент гр. 9382

\_\_\_\_\_

Докукин В. М.

Преподаватель

\_\_\_\_\_

Фирсов М. А.

Санкт-Петербург

2020

## ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Докукин В.М.

Группа 9382

Тема работы: АВЛ-деревья. Вставка и исключение.

Исходные данные:

Создание программы для генерации заданий с ответами к ним для проведения текущего контроля среди студентов. Задания и ответы должны выводиться в файл в удобной форме.

Содержание пояснительной записки:

«Аннотация», «Содержание», «Введение», «Исходное задание», «Выполнение задания», «Тестирование», «Заключение», «Список использованных материалов».

Предполагаемый объем пояснительной записки:

Не менее 30 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи реферата: 29.12.2020

Дата защиты реферата: 29.12.2020

Студент

Преподаватель

\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

Докукин В. М.

Фирсов М. А.

## **АННОТАЦИЯ**

Данная курсовая работа посвящена описанию процесса разработки программы, генерирующей задания для проведения текущего контроля среди студентов, её проектирования и тестирования. Здесь описаны классы, реализованные для выполнения поставленной задачи, их методы, отдельные функции и пользовательский интерфейс, а также алгоритмы, применяемые для решения как поставленной задачи, так и составляющих её подзадач. К курсовой работе приложено содержимое всех файлов, составляющих программу, названия и расширения этих файлов.

## **SUMMARY**

This course work is devoted to describing the process of developing a program to generate tasks used in students' current control, it's designing and testing. It describes classes implemented to accomplish the task, their methods, individual functions and user interface, as well as algorithms used to solve both the given task and its subtasks. The course work contains the contents of all files that make up the program, the files' names and extensions.

## СОДЕРЖАНИЕ

	Введение	5
1.	Исходное задание	6
2.	Выполнение задания	7
2.1.	Описание основных алгоритмов	7
2.2.	Классы и их методы	8
2.3.	Отдельные функции	11
2.4.	Пользовательский интерфейс	12
3.	Тестирование	17
	Заключение	23
	Список использованных источников	24
	Приложение А. Название приложения	25

## **ВВЕДЕНИЕ**

### **Цель работы.**

Разработка программы автоматической и/или ручной генерации заданий для проведения текущего контроля по теме «АВЛ-деревья, вставка и удаление элементов из АВЛ-дерева» и ответов к этим заданиям.

### **Задачи.**

1. Углубление познаний о языке C++.
2. Изучение структуры данных — бинарного дерева, её разновидностей (в частности, АВЛ-деревьев) и методов обработки.
3. Применение на практике изученных материалов путём написания программы.
4. Тестирование написанной программы.

### **Основные теоретические положения.**

Двоичное(бинарное) дерево поиска(БДП) — это бинарное дерево, на которое наложены дополнительные условия:

1. Поддеревья БДП — также БДП;
2. Ключи узлов левого поддерева БДП меньше или равны ключу корня БДП;
3. Ключи узлов правого поддерева БДП больше или равны ключу корня БДП.

Ключи узлов при этом должны быть сравнимы между собой.

АВЛ-дерево — это БДП, для каждого узла которого выполняется следующее условие: высота поддеревьев каждого узла различается не более чем на 1.

## ИСХОДНОЕ ЗАДАНИЕ

АВЛ-деревья - вставка и исключение. Текущий контроль.

"Текущий контроль" - создание программы для генерации заданий с ответами к ним для проведения текущего контроля среди студентов. Задания и ответы должны выводиться в файл в удобной форме: тексты заданий должны быть готовы для передачи студентам, проходящим ТК; все задания должны касаться конкретных экземпляров структуры данных (т.е. не должны быть вопросами по теории); ответы должны позволять удобную проверку правильности выполнения заданий.

## ВЫПОЛНЕНИЕ ЗАДАНИЯ

### 2.1. Описание основных алгоритмов

#### 1) Алгоритм вставки

Вставка в AVL-дерево производится следующим образом: ключ элемента для вставки последовательно сравнивается с элементами дерева, и в зависимости от того, меньше элемент или больше, выбирается направление спуска по дереву (лево или право соответственно). Спуск прекращается, когда находится пустой элемент по направлению спуска — на место этого элемента помещается вставляемый. После вставки производится балансировка дерева поворотами, если нарушается баланс.

Поворотов существует 4 типа:

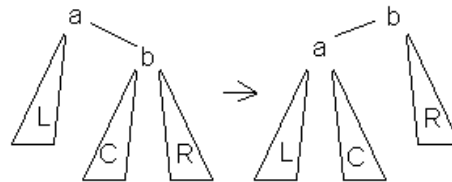
**Малый левый поворот** — используется, когда высота правого поддерева узла на 2 больше высоты левого поддерева узла, а высота левого поддерева правого сына узла меньше или равна высоте правого поддерева правого сына узла.

**Малый правый поворот** — используется, когда высота левого поддерева узла на 2 больше высоты правого поддерева узла, а высота правого поддерева левого сына узла меньше или равна высоте левого поддерева левого сына узла.

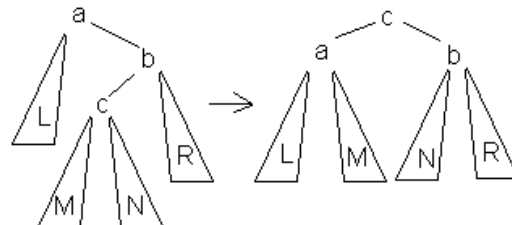
**Большой левый поворот** — используется, когда высота правого поддерева узла на 2 больше высоты левого поддерева узла, а высота левого поддерева правого сына узла больше высоты правого поддерева правого сына узла.

**Большой правый поворот** — используется, когда высота левого поддерева узла на 2 больше высоты правого поддерева узла, а высота правого поддерева левого сына узла больше высоты левого поддерева левого сына узла.

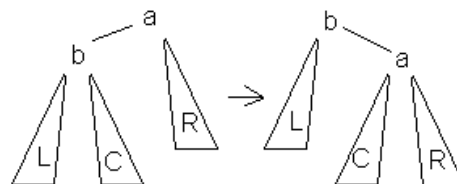
Малое левое вращение



Большое левое вращение



Малое правое вращение



Большое правое вращение

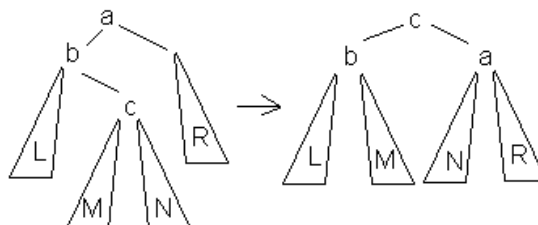


Иллюстрация 1 — виды поворотов в АВЛ-дереве.

## 2) Алгоритм удаления

Алгоритм удаления элемента из дерева практически идентичен алгоритму вставки. Если элемент — лист, он удаляется, и вызывается балансировка всех его предков в порядке от родителя к корню. Иначе ищется самая близкая по значению вершину в поддереве наибольшей высоты (правом или левом), которая перемещается на место удаляемой вершины.



## 2.2. Классы и их методы

В программе реализован класс `AVLTree`, объекты которого представляют собой АВЛ-дерево. Объект класса хранит в себе 4 приватных поля: `int height` – высота дерева; `int key` – ключ корня дерева; `AVLTree* left`, `AVLTree* right` – указатели на левый и правый поддеревья дерева соответственно.

Помимо этого, для работы с классом реализованы следующие методы:

1) `AVLTree(int key)` – конструктор класса. Создаёт узел дерева с пустыми поддеревьями и устанавливает его ключ равным аргументу конструктора.

2) `bool findKey(int k)` – рекурсивный метод, проверяющая, есть ли в данном дереве элемент с ключом, равным аргументу. Если ключ дерева равен искомому или дерево является пустым, рекурсия прекращается, иначе — вызывается `findKey()` для левого и правого поддерева. Метод возвращает `true`, если элемент с таким ключом найден, иначе — `false`.

3) `int getHeight()` – метод-геттер, предоставляет доступ к значению приватного поля `height` дерева.

4) `int bFactor()` – считает баланс-фактор дерева, т. е. разницу высот правого и левого поддеревьев. Возвращает баланс-фактор данного дерева путём вызова `getHeight()` для правого и левого поддеревьев.

5) `void fixHeight()` – метод, обновляющий высоту дерева. Для обновления выбирается максимальная из высот поддеревьев дерева, после чего поле `height` устанавливается в полученное значение + 1. Используется при поворотах.

6) `AVLTree* rotateLeftSmall(std::ostream* stream)` – метод малого поворота дерева влево. Метод меняет местами указатели на поддеревья дерева и его правого поддерева в соответствии с принципом малого левого поворота, изображённым на иллюстрации 1, а также записывает сообщение о повороте в поток `stream`. Возвращаемое значение метода — указатель на дерево.

7) `AVLTree* rotateLeftSmall(std::ostream* stream)` – метод малого поворота дерева вправо. Метод меняет местами указатели на поддеревья дерева и его левого поддерева в соответствии с принципом малого правого поворота,

изображённым на иллюстрации 1, а также записывает сообщение о повороте в поток stream. Возвращаемое значение метода — указатель на дерево.

**8) AVLTree\* balance(std::ostream\* stream)** – метод балансировки дерева. Метод обновляет высоту дерева и проверяет, нарушен ли баланс. Если баланс нарушен — производятся повороты в соответствии с отклонением баланса. В методе используется тот факт, что большой левый поворот является композицией малого правого и малого левого поворотов, а большой правый — малого левого и малого правого поворотов. Сообщения о поворотах записываются в поток stream. Метод возвращает указатель на дерево после балансировки.

**9) AVLTree\* insert(int key)** – вставляет в дерево элемент в соответствии с алгоритмом, описанным в 2.1. После вставки производится баланс дерева, если необходимо. Метод возвращает указатель на вставленный элемент после балансировки.

**10) AVLTree\* findMin()** - рекурсивный метод поиска минимального элемента. Согласно теоретической информации, минимальный элемент в АВЛ-дерева находится левее всех остальных. Таким образом, метод рекурсивно идёт влево по дереву, пока не найдёт элемент, у которого нет левого сына. Возвращаемое значение — указатель на минимальный элемент.

**11) AVLTree\* removeMin(std::ostream\* stream)** – рекурсивный метод, который удаляет элемент для замещения из поддереву в случае удаления узла, не являющегося листом. Согласно теоретической информации, при удалении элемента, если он не является листом, следует заменить его наиболее близким по значению элементом правого поддереву. Таким образом, метод ищет в правом поддереву минимальный элемент и удаляет его. Метод возвращает указатель на правое поддерево, если у дерева нет левых поддеревьев, либо указатель на дерево после балансировки.

**12) AVLTree\* remove(int k, std::ostream\* stream)** – метод, удаляющий элемент из дерева согласно алгоритму, описанному в 2.1. После удаления элемента производится балансировка дерева. Возвращает указатель на

**13)** `void destroy()` - рекурсивно удаляет дерево. Вызывает самого себя для левого и правого поддеревьев, после чего удаляет себя.

**14)** `void printTree(std::ostream* stream, int treeHeight, int depth = 0, int level = 0)` – рекурсивный метод печати дерева на экран. Выводит дерево на экран по уровням в удобном для восприятия виде.

### **2.3. Отдельные функции**

Для структуризации кода, упрощения его восприятия и модифицируемости написаны следующие вспомогательные функции:

**1)** `void treeOutput(std::ostream*, AVLTree* tree)` – функция-обертка для метода `printTree()` класса `AVLTree`. Написана с целью упростить код метода.

**2)** `void clear(AVLTree** trees, int* actions, int count)` – функция, очищающая память, выделенную под массивы деревьев `trees` и элементов для вставки/удаления `actions`.

**3)** `void saveTasks(AVLTree** trees, int* actions, int insertcount, int deletcount)` – функция, обеспечивающая сохранение результатов работы программы в файлы. Внутри функции открываются 2 потока вывода — `tasks` и `answers`, - привязанные к файлам `tasks.txt` и `answers.txt` соответственно. Тексты заданий выводятся в поток `tasks`, а ответы к ним — в поток `answers`.

**4)** `AVLTree* randGenerator()` - функция, генерирующая случайное дерево. Сначала генерируется количество узлов (от 5 до 20), а затем — элементы дерева. Функция возвращает указатель на корень сгенерированного дерева.

**5)** `AVLTree* userGenerator()` - функция, реализующая пользовательский ввод дерева. Внутри функции находится бесконечный цикл `while(true)`, в котором пользователь может вставлять элементы по ключу, удалять элементы по ключу, а также завершить редактирование дерева и сохранить его.

**6)** `int treeCreator(int mode)` – функция-обертка для функций `randGenerator()` и `userGenerator()`, реализующая пользовательский интерфейс. Внутри функции производится выбор количества заданий каждого вида, выделение памяти под массивы деревьев и условий заданий, а также показ итоговых заданий. Функция

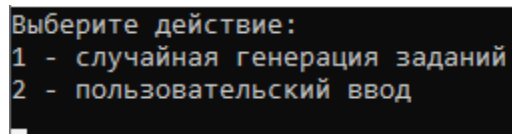
возвращает 0, если она завершилась успешно, или 1, если внутри функции произошла ошибка.

7) `int main()` - главная функция программы. В ней выбирается зерно генерации случайных чисел, а также выбор режима генерации заданий.

## 2.4. Пользовательский интерфейс

### 1) Стартовое меню

Происходит выбор пользователем режима работы программы.

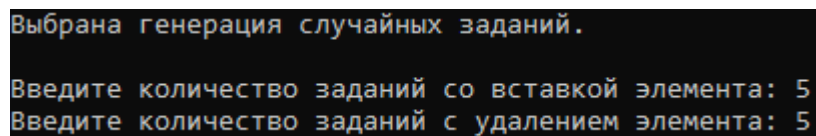


```
Выберите действие:  
1 - случайная генерация заданий  
2 - пользовательский ввод
```

Иллюстрация 2 — стартовое меню программы

### 2) Случайная генерация заданий

Происходит выбор пользователем количества заданий, а также генерация заданий.



```
Выбрана генерация случайных заданий.  
Введите количество заданий со вставкой элемента: 5  
Введите количество заданий с удалением элемента: 5
```

Иллюстрация 3 — ввод количества заданий

```

Промежуточные данные:
Производится вставка элемента 8
Производится вставка элемента 86
Производится вставка элемента 35
Производится малый поворот элемента 86 вправо.
Производится малый поворот элемента 8 влево.
Производится вставка элемента 29
Производится вставка элемента 82

Итоговое задание:
№2. Постройте дерево, получаемое из данного вставкой элемента 44.

      35
     /  \
    8    86
   /  \  /  \
  29  82

```

Иллюстрация 4 — случайная генерация заданий

### 3) Пользовательская генерация заданий.

Происходит ввод количества заданий каждого вида,

```

Выбран режим пользовательского ввода.

Введите количество заданий со вставкой элемента: 5
Введите количество заданий с удалением элемента: 5

```

Иллюстрация 5 — режим пользовательского ввода

после чего происходит ввод условия

```

Создание 1-го дерева для задания по вставке.
Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево
1
Введите элемент для вставки: 10
Производится вставка элемента 10
Ваше дерево:
10

Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево

```

Иллюстрация 6 — пользовательский ввод

с выводом промежуточных действий.

```
Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево
1
Введите элемент для вставки: 1000
Производится вставка элемента 1000
Производится малый поворот элемента 15 влево.
Ваше дерево:
      10
     /  \
    3    100
   / \   / \
  -5  15 1000
```

Иллюстрация 7 — промежуточные действия

По окончании ввода, вводится условие задания и выводится итоговое задание.

```
Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево
3
Ваше дерево сохранено.
Введите элемент, который необходимо вставить по условию задания: -1000

Итоговое задание:
№1. Постройте дерево, получаемое из данного вставкой элемента -1000.

      10
     /  \
    3    100
   / \   / \
  -5  15 1000

Создание 2-го дерева для задания по вставке.
Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево
```

Иллюстрация 8 — вывод условия

#### 4) Сохранение результатов работы в файлы

После ввода всех заданий или генерации случайным образом, программа предлагает пользователю сохранить результаты в файлы.

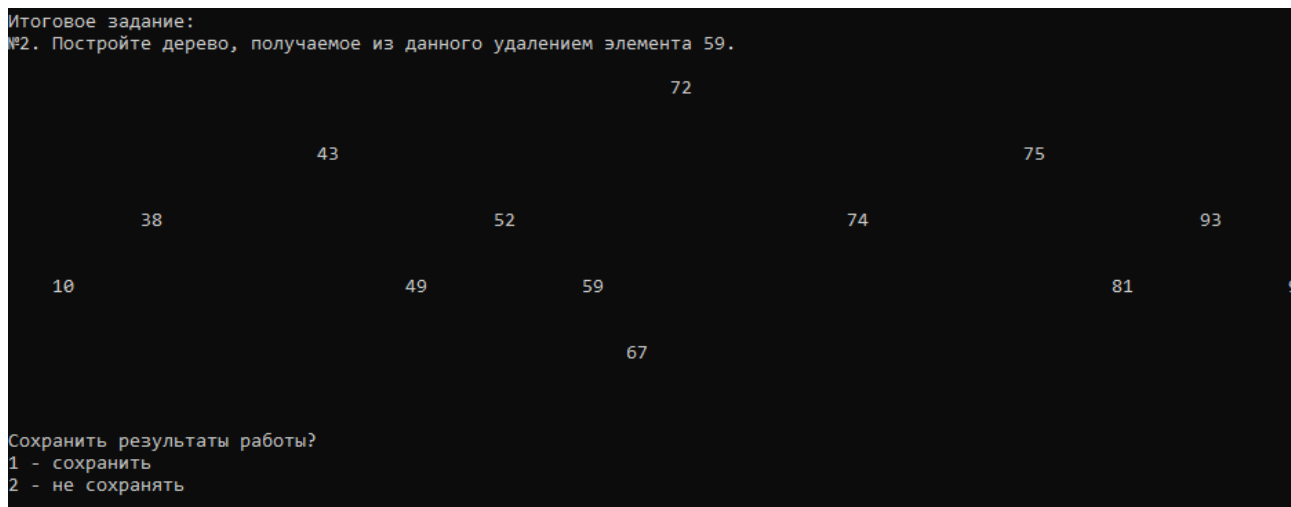


Иллюстрация 9 — запрос о сохранении в файл

При выборе первого варианта программа сохраняет результат в файлы и завершается, при выборе второго — возвращается в главное меню.

5) Примеры обработки некорректных вводов.

```
Выберите действие:  
1 - случайная генерация заданий  
2 - пользовательский ввод  
3  
Ошибка ввода.  
Выберите действие:  
1 - случайная генерация заданий  
2 - пользовательский ввод
```

Иллюстрация 10 — обработка некорректного ввода в главном меню.

```
Введите количество заданий со вставкой элемента: -3  
Введите количество заданий с удалением элемента: -5  
Ошибка при вводе количества заданий.  
Выберите действие:  
1 - случайная генерация заданий  
2 - пользовательский ввод
```

Иллюстрация 11 — обработка некорректного ввода при выборе количества заданий

```
Сохранить результаты работы?  
1 - сохранить  
2 - не сохранять  
5  
Ошибка ввода.  
Сохранить результаты работы?  
1 - сохранить  
2 - не сохранять
```

Иллюстрация 12 — обработка некорректного ввода при выборе опции сохранения в файлы



## ТЕСТИРОВАНИЕ

Результаты тестирования представлены на иллюстрациях ниже.

### 1) Вставка элементов в дерево

```
Введите элемент для вставки: 1
Производится вставка элемента 1
Ваше дерево:
1

Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево
2

Введите элемент для удаления: 2
Такого ключа в дереве нет!
Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево
1

Введите элемент для вставки: 2
Производится вставка элемента 2
Ваше дерево:
  1
    2

Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево
```

### 2) Удаление элемента из дерева

```
Ваше дерево:
  1
    2

Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево
2

Введите элемент для удаления: 2
Производится удаление элемента 2
Ваше дерево:
1

Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево
```

### 3) Малые повороты влево, вправо, большие повороты влево и вправо

```
Ваше дерево:
  1
   2

Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево
1
Введите элемент для вставки: 3
Производится вставка элемента 3
Производится малый поворот элемента 1 влево.
Ваше дерево:
  2
 1   3

Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево
```

```
Ваше дерево:
  2
 1   3
 0

Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево
1
Введите элемент для вставки: -1
Производится вставка элемента -1
Производится малый поворот элемента 1 вправо.
Ваше дерево:
  2
 0   3
-1  1

Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево
```

```

Ваше дерево:
          2
        /  \
       0    7
      /  \  /  \
     -1  1 3   10
          \   \
           8   15

Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево
1
Введите элемент для вставки: 9
Производится вставка элемента 9
Производится малый поворот элемента 10 вправо.
Производится малый поворот элемента 7 влево.
Ваше дерево:
          2
        /  \
       0    8
      /  \  /  \
     -1  1 7   10
          \   \
           3   9
              \
               15

```

```

Ваше дерево:
      50
     /  \
    25   75
   /  \  /  \
  12  30
      /  \
     40  50
    /  \  /  \
   12  40 75

```

#### 4) Удаление узла дерева, не являющегося листом

```
Ваше дерево:
      3
     / \
    2   7
   / \ / \
  0  5 5  9

Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево
2
Введите элемент для удаления: 3
Производится удаление элемента 3
Ваше дерево:
      5
     / \
    2   7
   / \ / \
  0  5 5  9

Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево
```

#### 5) Случайная генерация заданий

```
Промежуточные данные:
Производится вставка элемента 73
Производится вставка элемента 47
Производится вставка элемента 3
Производится малый поворот элемента 73 вправо.
Производится вставка элемента 57
Производится вставка элемента 82
Производится вставка элемента 11
Производится вставка элемента 19
Производится малый поворот элемента 3 влево.
Производится вставка элемента 60

Итоговое задание:
№1. Постройте дерево, получаемое из данного вставкой элемента 43.

      47
     / \
    11  73
   / \ / \
  3  19 57 82
   / \
  60
```

## 6) Пользовательская генерация заданий

```
Создание 1-го дерева для задания по вставке.
Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево
1
Введите элемент для вставки: 1
Производится вставка элемента 1
Ваше дерево:
1

Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево
2
Введите элемент для удаления: 2
Такого ключа в дереве нет!
Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево
1
Введите элемент для вставки: 2
Производится вставка элемента 2
Ваше дерево:
  1
    2

Выберите действие:
1 - вставка элемента
2 - удаление элемента
3 - сохранить дерево
3
Ваше дерево сохранено.
Введите элемент, который необходимо вставить по условию задания: 5

Итоговое задание:
№1. Постройте дерево, получаемое из данного вставкой элемента 5.

  1
    2
```

## 7) Сохранение заданий в файл

Текущий контроль  
Список заданий

№1. Постройте дерево, получаемое из данного вставкой элемента 62.



№2. Постройте дерево, получаемое из данного исключением элемента 20.



Текущий контроль  
Ответы к заданиям

№1.  
Операции над деревом:  
Производится вставка элемента 62.

Ответ:



№2.  
Операции над деревом:  
Производится удаление элемента 20.

Ответ:



## **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсовой работы было выполнено следующее:

1. Были изучены бинарные деревья поиска, их разновидности, методы и особенности работы с ними.
  2. Была написана программа, генерирующая задания для проведения текущего контроля по теме «АВЛ-деревья, вставка элементов в АВЛ-дерево и исключение элементов из АВЛ-дерева. Повороты».
  3. Было произведено тестирование программы, позволяющее качественно оценить работу программы и обработку критических ситуаций в программе.
- Исходный код программы размещён в приложении 1.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Википедия — свободная энциклопедия // АВЛ-дерево // <https://ru.wikipedia.org/wiki/АВЛ-дерево>
2. Wikipedia, the free encyclopedia // AVL tree // [https://en.wikipedia.org/wiki/AVL\\_tree](https://en.wikipedia.org/wiki/AVL_tree)
3. C++ Reference // C++ 11 standart // <https://en.cppreference.com/w/cpp/11>



## ПРИЛОЖЕНИЕ 1

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: AVLTree.h

```
#ifndef COURSEWORK_AVLTREE_H
#define COURSEWORK_AVLTREE_H

#include <iostream>
#include <fstream>

class AVLTree {
private:
    int height;
    int key;
    AVLTree* left;
    AVLTree* right;
public:
    AVLTree(int);
    int getHeight();
    bool findKey(int);
    void printTree(std::ostream*, int, int, int);
    int bFactor();
    void fixHeight();
    AVLTree* rotateRightSmall(std::ostream*);
    AVLTree* rotateLeftSmall(std::ostream*);
    AVLTree* balance(std::ostream*);
    AVLTree* insert(int, std::ostream*);
    AVLTree* findMin();
    AVLTree* removeMin(std::ostream*);
    AVLTree* remove(int, std::ostream*);
    void destroy();
};

#endif
```

Название файла: AVLTree.cpp

```
#include "pch.h"
#include <cmath>

#include "AVLTree.h"

AVLTree::AVLTree(int key) { // Конструктор дерева
    this->height = 1;
    this->key = key;
    this->left = nullptr;
    this->right = nullptr;
}
```

```

bool AVLTree::findKey(int k) { // Проверяем, есть ли в дереве
элемент с ключом == k
    if (this == nullptr) { // Если пустой элемент - прекращаем
рекурсию
        return false;
    }
    if (this->key == k) { // Если нашли - также прекращаем
рекурсию
        return true;
    }
    return this->left->findKey(k) | this->right->findKey(k); //
Запускаем поиск в левом и правом поддеревьях
}

int AVLTree::getHeight() { // Доступ к высоте элемента
    if (this != nullptr) {
        return this->height;
    }
    return 0;
}

int AVLTree::bFactor() { // Высчитывает баланс-фактор элемента
    return this->right->getHeight() - this->left->getHeight();
}

void AVLTree::fixHeight() { // Высчитываем высоту после поворота
    int lheight = left->getHeight();
    int rheight = right->getHeight();
    if (lheight > rheight) {
        this->height = lheight + 1;
    }
    else {
        this->height = rheight + 1;
    }
}

AVLTree* AVLTree::rotateRightSmall(std::ostream* stream) { //
Правый поворот
    if (stream != nullptr) *stream << "Производится малый поворот
элемента " << this->key << " вправо.\n";
    //std::cout << "Производится малый поворот элемента " <<
this->key << " вправо.\n";
    AVLTree* tmp = this->left;
    this->left = tmp->right;
    tmp->right = this;
    fixHeight();
    tmp->fixHeight();
    return tmp;
}

AVLTree* AVLTree::rotateLeftSmall(std::ostream* stream) { // Левый
поворот

```

```

        if (stream != nullptr) *stream << "Производится малый поворот
элемента " << this->key << " влево.\n";
        //std::cout << "Производится малый поворот элемента " <<
this->key << " влево.\n";
        AVLTree* tmp = right;
        this->right = tmp->left;
        tmp->left = this;
        fixHeight();
        tmp->fixHeight();
        return tmp;
    }

AVLTree* AVLTree::balance(std::ostream* stream) { //Балансируем
дерево
    fixHeight();
    if (this->bFactor() == 2)
    {
        if (this->right->bFactor() < 0) this->right = this-
>right->rotateRightSmall(stream);
        return this->rotateLeftSmall(stream);
    }
    if (this->bFactor() == -2)
    {
        if (this->left->bFactor() > 0) this->left = this->left-
>rotateLeftSmall(stream);
        return this->rotateRightSmall(stream);
    }
    return this;
}

AVLTree* AVLTree::insert(int k, std::ostream* stream) { //
Вставляем элемент с ключом == k
    if (this == nullptr) { // Если дошли до пустого элемента,
создаём новый элемент
        //std::cout << "Производится вставка элемента " << k <<
'\n';
        if (stream != nullptr) *stream << "Производится вставка
элемента " << k << '\n';
        return new AVLTree(k);
    }
    // Иначе...
    if (k < this->key) {
        this->left = this->left->insert(k, stream); // ...если
ключ элемента меньше ключа текущего, смотрим левое поддерево
    }
    if (k > this->key) {
        this->right = this->right->insert(k, stream); // ...если
ключ элемента больше ключа текущего, смотрим правое поддерево
    }
    return balance(stream);
}

AVLTree* AVLTree::findMin() { // Возвращает минимальный элемент

```

```

        if (this->left != nullptr) {
            return this->left->findMin();
        }
        else if (this->right != nullptr) {
            return this->right->findMin();
        }
        return this;
    }

AVLTree* AVLTree::removeMin(std::ostream* stream) { // Удаляет
МИНИМАЛЬНЫЙ ЭЛЕМЕНТ
    if (left == nullptr) {
        return right;
    }
    left = left->removeMin(stream);
    return balance(stream);
}

AVLTree* AVLTree::remove(int k, std::ostream* stream) { // Удаляет
элемент с ключом == k
    if (this == nullptr) return this; // Если дошли до пустого
элемента - возвращаем 0
    if (k < this->key) this->left = this->left->remove(k,
stream); // Если ключ меньше ключа текущего элемента, ищем в левом
поддереве
    else if (k > this->key) this->right = this->right->remove(k,
stream); // Если ключ больше ключа текущего элемента, ищем в
правом поддереве
    else {
        AVLTree* ltree = this->left;
        AVLTree* rtree = this->right;
        //std::cout << "Производится удаление элемента " << k <<
'\n';
        if (stream != nullptr) *stream << "Производится удаление
элемента " << k << '\n';
        delete this;
        if (!rtree) return ltree;
        AVLTree* min = rtree->findMin();
        min->right = rtree->removeMin(stream);
        min->left = ltree;
        return min->balance(stream);
    }
    return balance(stream);
}

void AVLTree::destroy() { // Освобождаем память, выделенную под
дерево
    if (this != nullptr) {
        this->left->destroy();
        this->right->destroy();
        delete this;
    }
}

```

```

void AVLTree::printTree(std::ostream* stream, int treeHeight, int
depth = 0, int level = 0) { // Печатаем дерево в std::cout и в
файл по уровням
    if (depth == level) {
        if (this != nullptr) *stream << key; // Печатаем корень
        else *stream << " ";
        for (int i = 0; i < pow(2, treeHeight - (level + 1));
i++) *stream << "\t"; //отступ
        if (level < treeHeight - 1) *stream << " ";
        return;
    }
    else {
        if (this == nullptr) {
            this->printTree(stream, treeHeight, depth + 1,
level);
            this->printTree(stream, treeHeight, depth + 1,
level);
        }
        else{
            this->left->printTree(stream, treeHeight, depth + 1,
level); // Печатаем левое поддерево
            this->right->printTree(stream, treeHeight, depth + 1,
level); // Печатаем левое поддерево
        }
    }
}
}

```

**Название файла: main.cpp**

```

#include "pch.h"
#include <iostream>
#include <string>
#include <ctime>

#include "AVLTree.h"

void treeOutput(std::ostream* stream, AVLTree* tree) {
    if (tree == nullptr) {
        *stream << "[пустое дерево]\n";
        return;
    }
    for (int j = 0; j < tree->getHeight(); j++) {
        if (j < tree->getHeight() - 1) {
            for (int k = 0; k < pow(2, tree->getHeight() - j -
2) - 1; k++) {
                *stream << "\t";
            }
            *stream << " ";
        }
        tree->printTree(stream, tree->getHeight(), 0, j);
        *stream << "\n\n";
    }
}

```

```

    }
}

void clear(AVLTree** trees, int* actions, int count) { // Очищаем
все деревья
    for (int i = 0; i < count; i++) {
        trees[i]->destroy();
    }
    delete actions;
    delete trees;
}

void saveTasks(AVLTree** trees, int* actions, int insertcount, int
deletecount) { // Заполнение файла с заданиями
    std::string insstr = ". Постройте дерево, получаемое из
данного вставкой элемента ";
    std::string delstr = ". Постройте дерево, получаемое из
данного исключением элемента ";

    std::filebuf t;
    t.open("tasks.txt", std::ios::out);
    std::ostream tasks(&t);
    tasks << "Текущий контроль\nСписок заданий\n\n";

    for (int i = 0; i < insertcount; i++) {
        tasks << "№" << i + 1 << insstr << actions[i] <<
".\n\n"; // Записываем условие
        treeOutput(&tasks, trees[i]);
        tasks << '\n';
    }
    for (int i = insertcount; i < deletecount + insertcount; i++)
    {
        tasks << "№" << i + 1 << delstr << actions[i] <<
".\n\n"; // Записываем условие
        treeOutput(&tasks, trees[i]);
        tasks << '\n';
    }
    t.close();

    std::filebuf a;
    a.open("answers.txt", std::ios::out);
    std::ostream answers(&a);
    answers << "Текущий контроль\nОтветы к заданиям\n\n";
    for (int i = 0; i < insertcount; i++) {
        answers << "№" << i + 1 << ".\nОперации над деревом:\n";
        trees[i]->insert(actions[i], &answers); // Вставляем в
дерево элемент
        answers << "\nОтвет:\n";
        treeOutput(&answers, trees[i]); // Печатаем полученное
дерево
        answers << '\n';
    }
}

```

```

        for (int i = insertcount; i < deletecount + insertcount; i++)
        {
            answers << "№" << i + 1 << ".\nОперации над деревом:\n";
            trees[i] = trees[i]->remove(actions[i], &answers); //
Удаляем элемент из дерева
            answers << "\nОтвет:\n";
            treeOutput(&answers, trees[i]); // Печатаем полученное
дерево
            answers << '\n';
        }
        std::cout << '\n';
        a.close();
    }

AVLTree* randGenerator() {
    int key;
    int count = rand() % 16 + 5; // Генерируем количество
элементов
    AVLTree* randtree = nullptr;
    for (int i = 0; i < count; i++) {
        key = rand() % 100 + 1;
        while (randtree->findKey(key)) {
            key = rand() % 100 + 1; // Если такой элемент уже
есть - генерируем заново
        }
        randtree = randtree->insert(key, &std::cout); //
Вставляем элемент в дерево
    }
    return randtree;
}

AVLTree* userGenerator() {
    int key, option;
    AVLTree* usertree = nullptr;
    while (true) { // Заполняем дерево
        std::cout << "Выберите действие:\n1 - вставка
элемента\n2 - удаление элемента\n3 - сохранить дерево\n";
        std::cin >> option;
        switch (option) {
            case 1:
                std::cout << "Введите элемент для вставки: ";
                std::cin >> key;
                if (!usertree->findKey(key)) usertree = usertree-
>insert(key, &std::cout);
                else {
                    std::cout << "Такой ключ уже имеется в
дереве!\n";
                    break;
                }
                std::cout << "Ваше дерево:\n";
                treeOutput(&std::cout, usertree);
                break;
            case 2:

```

```

        std::cout << "Введите элемент для удаления: ";
        std::cin >> key;
        if (usertree->findKey(key)) usertree = usertree-
>remove(key, &std::cout);
        else {
            std::cout << "Такого ключа в дереве нет!\n";
            break;
        }
        std::cout << "Ваше дерево:\n";
        treeOutput(&std::cout, usertree);
        break;
    case 3:
        std::cout << "Ваше дерево сохранено.\n";
        return usertree;
        break;
    default:
        std::cout << "Ошибка ввода.\n";
        break;
    }
}

int treeCreator(int mode) {
    int insertcount;
    std::cout << "Введите количество заданий со вставкой
элемента: ";
    std::cin >> insertcount;

    int deletecount;
    std::cout << "Введите количество заданий с удалением
элемента: ";
    std::cin >> deletecount;

    if ((insertcount < 0) && (deletecount < 0)) { // Если введено
некорректное количество - возвращаемся в главное меню
        std::cout << "Ошибка при вводе количества заданий.\n";
        return 1;
    }

    int count = insertcount + deletecount;
    AVLTree** trees = new AVLTree*[count];
    int* actions = new int[count];
    int option;

    std::cout << "-----\n";
    for (int i = 0; i < insertcount; i++) {
        if (mode == 1) {
            std::cout << "Промежуточные данные:\n";
            trees[i] = randGenerator(); // Генерируем рандомные
деревья

            actions[i] = rand() % 100 + 1;
            while (trees[i]->findKey(actions[i])) {

```



```

        actions[i] = rand() % 100 + 1; // Генерируем
число, которое нужно будет вставить по заданию
    }
}
if (mode == 2) {
    std::cout << "Создание " << i + 1 << "-го дерева
для задания по вставке.\n";
    trees[i] = userGenerator();
    std::cout << "Введите элемент, который необходимо
вставить по условию задания: ";
    while (true) {
        std::cin >> actions[i];
        if (!trees[i]->findKey(actions[i])) break;
        std::cout << "Некорректный ключ для вставки.";
    }
}
std::cout << "\nИтоговое задание:\n№" << i + 1 << ".
Постройте дерево, получаемое из данного вставкой элемента " <<
actions[i] << ".\n\n";
treeOutput(&std::cout, trees[i]);
std::cout << '\n';
}
for (int i = insertcount; i < count; i++) {
    if (mode == 1) {
        std::cout << "Промежуточные данные:\n";
        trees[i] = randGenerator(); // Генерируем рандомные
деревья
        actions[i] = rand() % 100 + 1;
        while (!trees[i]->findKey(actions[i])) {
            actions[i] = rand() % 100 + 1; // Генерируем
число, которое нужно будет удалить по заданию
        }
    }
    if (mode == 2) {
        std::cout << "Создание " << i + 1 << "-го дерева
для задания по удалению.\n";
        trees[i] = userGenerator();
        std::cout << "Введите элемент, который необходимо
удалить по условию задания: ";
        while (true) {
            std::cin >> actions[i];
            if (trees[i]->findKey(actions[i])) break;
            std::cout << "Некорректный ключ для
удаления.";
        }
    }
    std::cout << "\nИтоговое задание:\n№" << i + 1 << ".
Постройте дерево, получаемое из данного удалением элемента " <<
actions[i] << ".\n\n";
    treeOutput(&std::cout, trees[i]);
    std::cout << '\n';
}
}

```

```

        while (true) {
            std::cout << "Сохранить результаты работы?\n1 -
сохранить\n2 - не сохранять\n";
            std::cin >> option;
            switch (option) {
                case 1:
                    saveTasks(trees, actions, insertcount,
deletecount);
                    clear(trees, actions, count);
                    return 0;
                    break;
                case 2:
                    clear(trees, actions, count);
                    return 1;
                    break;
                default:
                    std::cout << "Ошибка ввода.\n";
                    break;
            }
        }
    }

int main()
{
    srand(time(0)); // Берём случайное зерно генерации
    setlocale(LC_ALL, "rus");

    int option = 0;
    while (true) { // Основной цикл программы
        std::cout << "Выберите действие:\n1 - случайная
генерация заданий\n2 - пользовательский ввод\n";
        std::cin >> option;
        switch (option) {
            case 1:
                std::cout << "Выбрана генерация случайных
заданий.\n\n";
                if (!treeCreator(1)) {
                    return 0;
                }
                break;
            case 2:
                std::cout << "Выбран режим пользовательского
ввода.\n\n";
                if (!treeCreator(2)) {
                    return 0;
                }
                break;
            default:
                std::cout << "Ошибка ввода.\n";
        }
    }
}

```