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

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Алгоритмы и структуры данных»

Тема: АВЛ-деревья.

Студент гр. 9381 _____ Птичкин С. А.

Преподаватель _____ Фирсов М. А.

Санкт-Петербург

2020

Цель работы.

Изучить структуру АВЛ-деревьев. Написать класс АВЛ-дерева. Написать программу взаимодействия с этой структурой данных.

Задание.

Вариант 17. БДП: АВЛ-дерево, действие 1+2в.

1) По заданной последовательности элементов Elem построить структуру данных определённого типа – БДП или хеш-таблицу;
2) Выполнить действие: записать в файл элементы построенного БДП в порядке их возрастания; вывести построенное БДП на экран в наглядном виде.

Выполнение работы.

Для применения некоторых функций были подключены заголовочные файлы iostream, string, fstream.

Описание алгоритма.

Алгоритм добавления элемента:

Вставка нового элемента в АВЛ-дерево выполняется так же, как это делается в простых деревьях поиска: спускаемся вниз по дереву, выбирая правое или левое направление движения в зависимости от результата сравнения ключа в текущем узле и вставляемого ключа. Единственное отличие заключается в том, что при возвращении из рекурсии (т.е. после того, как элемент добавлен либо в правое, либо в левое поддерево, и это дерево сбалансировано) выполняется балансировка текущего узла.

Балансировка дерева происходит, когда разница между высотами

поддеревьев одного элемента становится равной 2. В таком случае, в зависимости от конфигурации, необходимо произвести серию вращений, как это показано на рис.1.

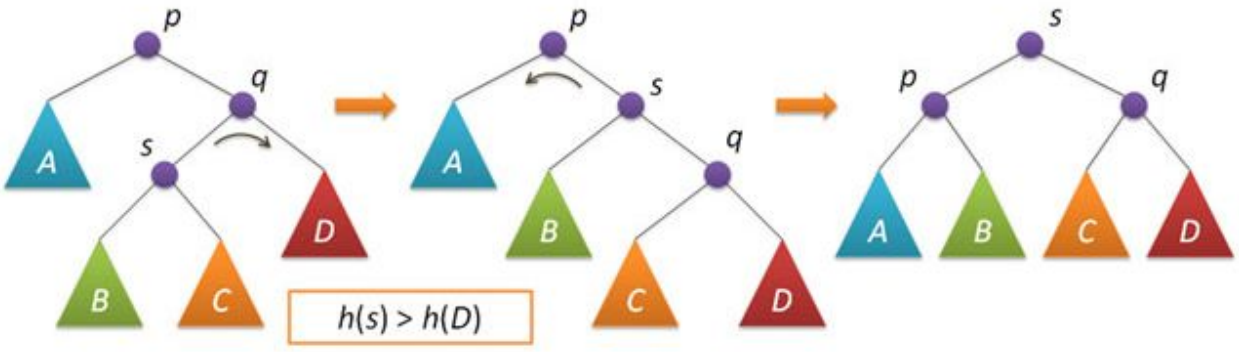


Рис.1 Балансировка дерева с помощью вращений.

Алгоритм исключения минимального элемента:

Начинаем рекурсивный поиск элемента в левых поддеревьях. Как только встречаем нулевой указатель в левом поддереве, сохраняем указатель на правое поддерево, удаляем сам элемент и возвращаем указатель на правое поддерево. При каждом выходе из рекурсии ребалансируем дерево.

Структура программы.

Программа разбита на 3 файла: AVL_Tree.h, хранящий определение класса. AVL_Tree.cpp, хранящий реализацию методов для работы с АВЛ-деревьями. AiSD_lb5.cpp, в которой реализован пользовательский интерфейс, обработка данных, функции консольного и файлового ввода/вывода.

Пользовательские типы данных.

Класс AVL_Tree.

Объекты данного класса представляют собой AVL-дерево. У объекта есть приватные поля: int height, хранящий высоту дерева, int key, хранящий значение

ключа, AVL-Tree* left и AVL-Tree* right, хранящие указатели на поддеревья. Также у класса определены некоторые методы для работы с ним.

Методы класса AVL-Tree:

1) **AVL_Tree::AVL_Tree(int key)**

   Конструктор класса AVL-дерева. Получает на вход значение ключа k, поле height инициализируется единицей, поля left и right - нулевыми указателями.

2) **bool AVL_Tree::find_key(int k)**

   Данный метод предназначен для проверки наличия в дереве элемента с заданным ключём. На вход передаётся сам ключ k. Функция рекурсивно обходит всё дерево и возвращает true при нахождении, false при отсутствии.

3) **int AVL_Tree::get_height()**

   Данный метод предназначен для получения значения из поля height. Метод ничего не принимает и возвращает поле height объекта класса.

4) **int AVL_Tree::get_key()**

   Данный метод предназначен для получения значения из поля key. Метод ничего не принимает и возвращает поле key объекта класса.

5) **int AVL_Tree::bfactor()**

   Данный метод возвращает разницу между высотами поддеревьев у элемента дерева. Метод ничего не принимает на вход.

6) **void AVL_Tree::fixheight()**

   Данный метод предназначен для обновления высоты у каждого элемента дерева. Метод ничего не принимает на вход и ничего не возвращает.

7) **AVL_Tree* AVL_Tree::rotateright(ostream* stream)**

   Данный метод производит поворот вправо. На вход подаётся адрес потока

вывода stream, в который выводится промежуточная информация, по умолчанию значение - nullptr. Метод возвращает указатель на элемент, полученный после поворота.

8) **AVL_Tree* AVL_Tree::rotateleft(ostream* stream)**

Данный метод производит поворот влево. На вход подаётся адрес потока вывода stream, в который выводится промежуточная информация, по умолчанию значение - nullptr. Метод возвращает указатель на элемент, полученный после поворота.

9) **AVL_Tree* AVL_Tree::balance(ostream* stream)**

Данный метод предназначен для выравнивания дерева по высоте после вставки/исключения элемента. На вход подаётся адрес потока вывода stream, которая в дальнейшем передаётся в методы левого и правого поворота. Функция возвращает указатель на элемент, относительно которого производилась балансировка.

10) **AVL_Tree* AVL_Tree::insert(int k, ostream* stream)**

Метод предназначен для вставки элемента в дерево. Метод принимает на вход ключ k и в зависимости от того меньше или больше он, чем значение текущего элемента вызывает рекурсивно этот же метод для своего левого или правого поддерева. Также на вход подаётся адрес потока вывода stream, в который выводится промежуточная информация, по умолчанию значение - nullptr. Дойдя до нулевого указателя происходит создание нового элемента. Функция сначала возвращает указатель на новый элемент, а затем возвращаемое значение функции балансировки.

11) **AVL_Tree* AVL_Tree::findmin()**

Данный метод рекурсивно обходит дерево и возвращает указатель на минимальный элемент дерева. Функция ничего не принимает на вход.

12) AVL_Tree* AVL_Tree::removemin()

Данный метод находит минимальный элемент дерева и удаляет его. Входные параметры отсутствуют. Метод возвращает указатель на правое поддерево.

13) AVL_Tree* AVL_Tree::remove(int k, ostream* stream)

Данный метод предназначен для удаления из дерева элемента с ключём k. Метод принимает на вход сам ключ k и адрес потока вывода stream, в который выводится промежуточная информация, по умолчанию значение - nullptr. Метод возвращает возвращаемое значение метода балансировки.

14) void AVL_Tree::destroy()

Метод предназначен для рекурсивного удаления дерева. Функция проходит всё дерево и очищает динамически выделенную память под элементы. Метод ничего не принимает и ничего не возвращает.

15) void AVL_Tree::print_tree(ostream* stream, int depth)

Данный метод предназначен для печати дерева в переданный поток ввода. Функция принимает указатель на поток stream и глубину рекурсивного погружения, которая отражает смещение элементов по высоте. Метод ничего не возвращает.

Основные функции.

1) Функция main. int main()

Функция не принимает никаких параметров. Данная функция предназначена для стартового диалога с пользователем. Здесь идёт выбор ввода данных, либо из консоли, либо из файла. За корректность введённых данных отвечает функция input_num. Ввод команды 1 вызывает функцию консольного ввода, команда 2 - файлового. Возвращаемые значения этих функций

определяют, будет ли цикл продолжаться с возможностью ввести новые данные, либо программа завершится.

2) Функция file_input. int file_input()

Функция не принимает никаких параметров. В начале объявляются все необходимые для работы переменные, выделяется память под имя файла. Затем считывается имя файла, и файл открывается при корректном имени. После этого из файла последовательно считываются целочисленные ключи и внедряются в созданное дерево. При этом вычисляется их количество. Затем файл закрывается и вызывается work_with_data, функция работы с деревом. Функция возвращает то же значение, что и work_with_data.

3) Функция console_input. int console_input()

Функция не принимает никаких параметров. В начале объявляются все необходимые для работы переменные. Затем пользователь вводит количество элементов в дереве. После чего из консоли считываются целочисленные ключи и внедряются в созданное дерево. Затем файл закрывается и вызывается work_with_data, функция работы с деревом. Функция возвращает то же значение, что и work_with_data.

4) int work_with_data(AVL_Tree* input_tree, int count_of_elements)

Функция принимает на вход созданное АВЛ-дерево и количество элементов в нём. Сначала дерево выводится на экран. Затем создаётся массив интов mass_rise_keys. В цикле массив заполняется минимальными ключами из дерева, при этом данный элемент сразу удаляется, пока дерево не опустеет. Затем полученный массив выводится на экран. Далее пользователь попадает в меню, где выбирает сохранить ли полученную последовательность элементов в файл, продолжить или завершить программу. При записи вызывается функция

output_tree_elem, с массивом mass_rise_keys в качестве параметра. Функция возвращает 1 для продолжения работы, 0 - для завершения.

5) void output_tree_elem(int* mass_rise_keys, int count_of_elements)

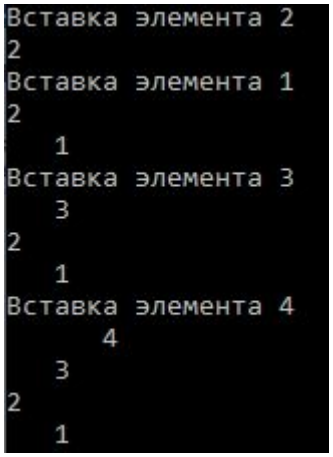
Функция принимает на вход массив элементов в порядке возрастания и их количество. Сначала пользователю предлагается назвать имя файла для записи. Затем файл открывается и в него последовательно записываются элементы массива. Файл закрывается и функция ничего не возвращает.

6) Функция input_num. int input_num(string message)

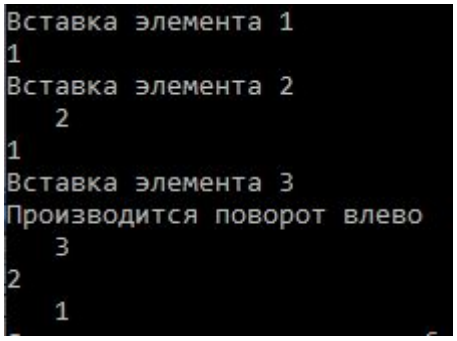
Функция принимает на вход сообщение, выводимое пользователю. Функция предназначена для корректного считывания числа из потока ввода. На вход принимается адрес строки с сообщением пользователю, что ему делать. Объявляется переменная для записи числа и выделяется буфер на 10 символов. Затем из cin считывается 10 символов в буфер. Далее в цикле из данного буфера считывается число функцией sscanf. Пока функция не вернёт 1 - количество верно считанных аргументов, ввод не прекратится. Когда наконец число считается, оно возвращается функцией. Память, выделенная под буфер очищается.

Тестирование. Работа с АВЛ-деревом.

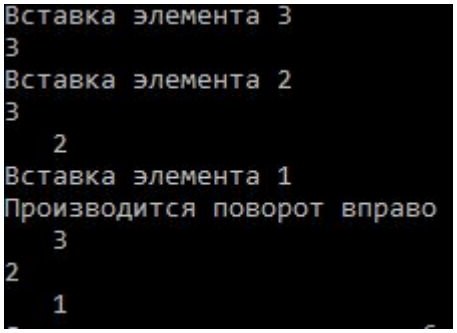
1) Вставка элементов.



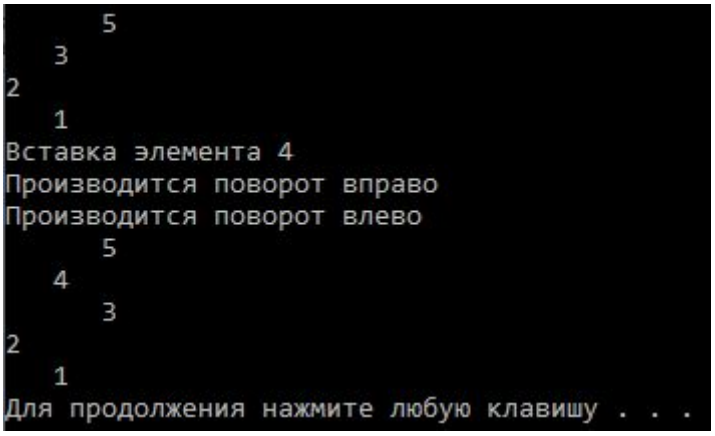
2) Поворот влево.



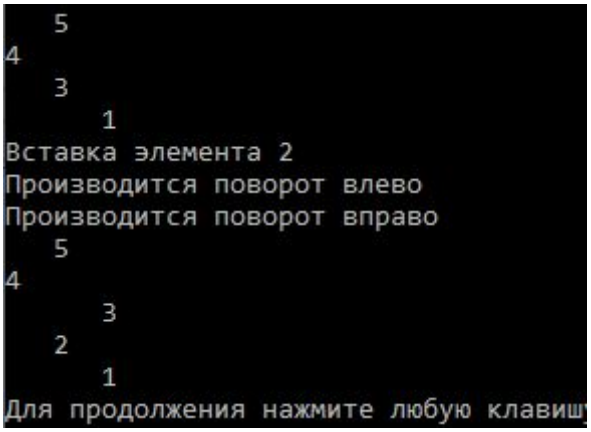
3) Поворот вправо.



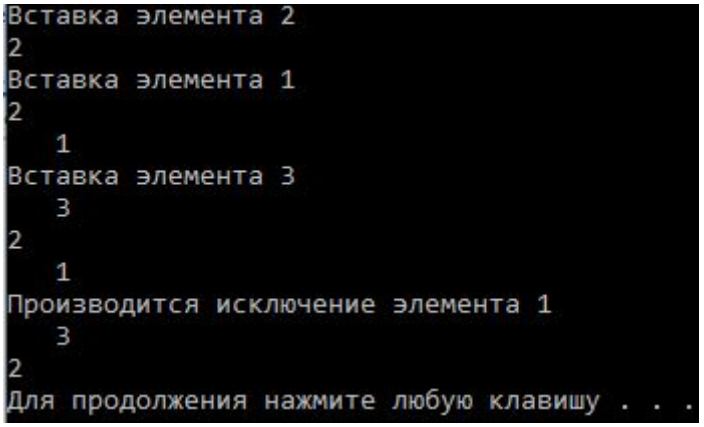
4) Двойной поворот(RL).



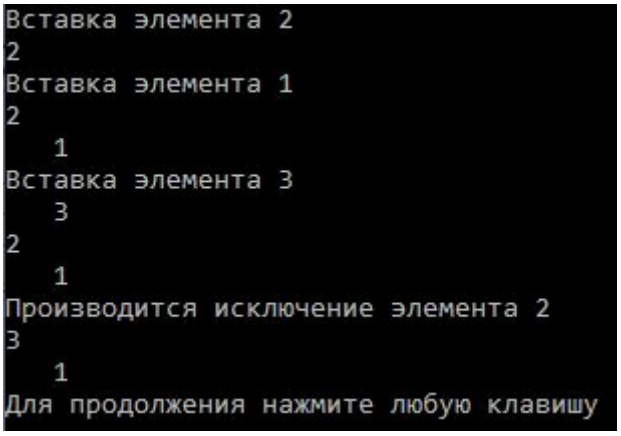
5) Двойной поворот(LR).



6) Удаление листа.



7) Удаление корня.



Тестирование. Вывод элементов дерева в порядке возрастания.

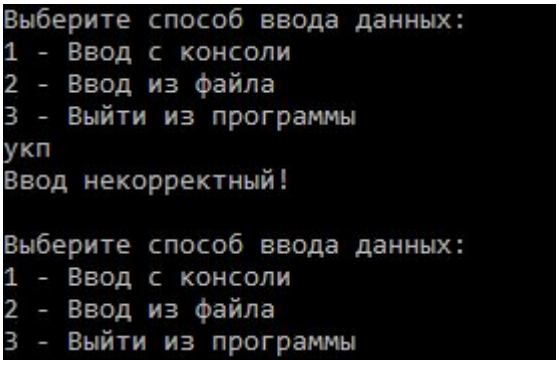
Результаты теста входных данных представлены в таблице 1.

Таблица 1- Результаты тестирования

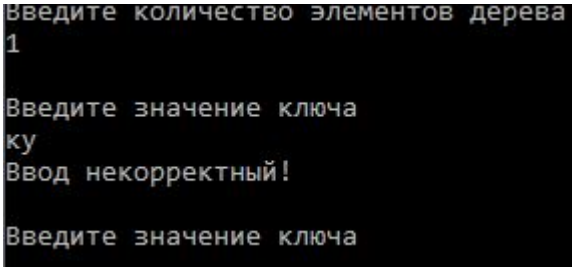
№ Теста	Входные данные	Выходные данные
1	2 1 3	1 2 3
2	4 5 1 2 3 6 9 0	0 1 2 3 4 5 6 9
3		
4	-4 -100 100 1 2 0 3 6	-100 -4 0 1 2 3 6 100

Обработка исключительных ситуаций.

1) **Некорректные действия**



2) Ввод с консоли



3) Некорректная команда

```
Выберите дальнейшее действие:
1 - вывести элементы дерева в порядке возрастания в файл и продолжить
2 - вывести элементы дерева в порядке возрастания в файл и выйти
3 - продолжить без вывода в файл
4 - выйти из программы
5
Команда не распознана!

Выберите дальнейшее действие:
1 - вывести элементы дерева в порядке возрастания в файл и продолжить
2 - вывести элементы дерева в порядке возрастания в файл и выйти
3 - продолжить без вывода в файл
4 - выйти из программы
```

4) Некорректное имя файла

```
Введите имя файла

уцкцук3

Файла с таким именем не найдено!
Введите имя файла
```

Вывод.

Была написана программа с использованием иерархических списков и рекурсивных алгоритмов. Были изучены принципы работы с иерархическими списками, взаимодействие с ними при помощи рекурсии, их применение для решения задач.

Приложение А

Исходный код программы

Название файла: AVL_Tree.h

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

class AVL_Tree {
private:
	int height;
	int key;
	AVL_Tree* left;
	AVL_Tree* right;
public:
	void destroy();
	AVL_Tree(int key);
	bool find_key(int k);
	void print_tree(ostream* stream, int depth = 0);
	int get_height();
	int get_key();
	int bfactor();
	void fixheight();
	AVL_Tree* rotateright(ostream* stream = nullptr);
	AVL_Tree* rotateleft(ostream* stream = nullptr);
	AVL_Tree* balance(ostream* stream = nullptr);
	AVL_Tree* insert(int k, ostream* stream = nullptr);
	AVL_Tree* findmin();
	AVL_Tree* removemin();
	AVL_Tree* remove(int k, ostream* stream = nullptr);
};
```

Название файла: AVL_Tree.cpp:

```
#include "AVL_Tree.h"

AVL_Tree::AVL_Tree(int key) {//конструктор
	this->height = 1;
```

```
	this->key = key;
	left = nullptr;
	right = nullptr;
}

bool AVL_Tree::find_key(int k) { //метод нахождения элемента с ключём k
	if (this == nullptr) {
		return 0;
	}
	if (key == k) {
		return true;
	}
	if ((left->find_key(k)) || (right->find_key(k))) {//рекурсивный вызов
для поддеревьев
		return true;
	}
	return false;
}

int AVL_Tree::get_height() {//возвращает значение высоты дерева
	if (this != nullptr) {
		return height;
	}
	return 0;
}

int AVL_Tree::get_key() { //возвращает ключ текущего элемента
	if (this != nullptr) {
		return key;
	}
	return 0;
}

int AVL_Tree::bfactor() { //разница между высотами поддеревьев
	return right->get_height() - left->get_height();
}
```

```
void AVL_Tree::fixheight() {  //обновляет высоту после вставки/удаления
элемента
	int hl = left->get_height();
	int hr = right->get_height();
	if (hl > hr) {
		height = hl + 1;
	}
	else {
		height = hr + 1;
	}
}

AVL_Tree* AVL_Tree::rotateright(ostream* stream) {//поворот вправо
	if (stream != nullptr)
		*stream << "Производится поворот вправо\n";
	cout << "Производится поворот вправо\n";
	AVL_Tree* q = left;
	left = q->right;
	q->right = this;
	fixheight();
	q->fixheight();
	return q;
}

AVL_Tree* AVL_Tree::rotateleft(ostream* stream) {//поворот влево
	if (stream != nullptr)
		*stream << "Производится поворот влево\n";
	cout << "Производится поворот влево\n";
	AVL_Tree* q = right;
	right = q->left;
	q->left = this;
	fixheight();
	q->fixheight();
	return q;
}
```

```
AVL_Tree* AVL_Tree::balance(ostream* stream) {//ребалансировка дерева
	fixheight();
	if (bfactor() == 2)
	{
		if (right->bfactor() < 0)
			right = right->rotateright(stream);
		return rotateleft(stream);
	}
	if (bfactor() == -2)
	{
		if (left->bfactor() > 0)
			left = left->rotateleft(stream);
		return rotateright(stream);
	}
	return this;
}

AVL_Tree* AVL_Tree::insert(int k, ostream* stream) {//вставка элемента с
ключём k
	if (!this) {
		cout << "Вставка элемента " << k << '\n';
		if (stream != nullptr)
			*stream << "Вставка элемента " << k << '\n';
		return new AVL_Tree(k);
	}
	if (k < key) {
		left = left->insert(k, stream); //рекурсивный вызов для
поддеревьев
	}
	if (k > key) {
		right = right->insert(k, stream);//рекурсивный вызов для
поддеревьев
	}
	return balance(stream);
}
```

```
AVL_Tree* AVL_Tree::findmin() { //нахождение элемента с минимальным ключём
	if (left != nullptr) {
		return left->findmin();
	}
	return this;
}

AVL_Tree* AVL_Tree::removemin() {  //удаление элемента с минимальным ключём
	if (left == nullptr) {
		AVL_Tree* r = right;
		cout << "Удаление элемента "<< key<<'\n';
		delete this;
		return r;
	}
	left = left->removemin();
	return balance();
}

AVL_Tree* AVL_Tree::remove(int k, ostream* stream) { //исключение элемента с
ключём k
	if (!this) return nullptr;
	if (k < key)
		left = left->remove(k, stream);//рекурсивный вызов для
поддеревьев
	else if (k > key)
		right = right->remove(k, stream);//рекурсивный вызов для
поддеревьев
	else
	{
		AVL_Tree* q = left;
		AVL_Tree* r = right;
		cout << "Производится исключение элемента " << k << '\n';
		if (stream != nullptr)
			*stream << "Производится исключение элемента " << k <<
'\n';
```

```
            delete this;
            if (!r) return q;
            AVL_Tree* min = r->findmin();
            min->right = r->removemin();
            min->left = q;
            return min->balance(stream);
      }
      return balance(stream);
}

void AVL_Tree::destroy() {//рекурсивная очистка дерева
      if (this != nullptr) {
            left->destroy();
            right->destroy();
            delete this;
      }
}

void AVL_Tree::print_tree(ostream* stream, int depth) { //печать дерева в
поток
      if (this != nullptr) {
            if (right != nullptr) {
                  right->print_tree(stream, depth + 1);//рекурсивный вызов
для поддеревьев
            }
            for (int i = 0; i < depth; i++) *stream << "   ";//число пробелов
зависит от высоты
            *stream << key << endl;
            if (left != nullptr) {
                  left->print_tree(stream, depth + 1);//рекурсивный вызов для
поддеревьев
            }
      }
      else {
            *stream << "Пустое дерево\n";
      }
```

```
}
```

Файл AiSD_lb5.cpp:

```
#include "AVL_Tree.h"
#include <string>

int input_num(string message) {
	int num = 0;
	cout << message << '\n';
	char* input = new char[10];
	fgets(input, 10, stdin);
	while (sscanf_s(input, "%d", &num) != 1) {
		cout << "Ввод некорректный!\n" << message << '\n';
		fgets(input, 10, stdin);
	}
	delete[] input;
	return num;
}

void output_tree_elem(int* mass_rise_keys, int count_of_elements) {
	char* file_name = new char[256];
	cout << "Введите имя файла сохранения\n";
	cin >> file_name;
	getchar(); //вытаскиваем символ переноса строки из потока
	fstream output_file;
	AVL_Tree* min_elem = nullptr;
	output_file.open(file_name, fstream::out | fstream::app);//открытие
или создание файла на запись
	cout << "Промежуточные операции в ходе выгрузки элементов в файл:\n";
	for (int i = 0; i < count_of_elements; i++) {
		output_file << mass_rise_keys[i]<<' '; //записываем минимальный
элемент в файл
	}
	output_file << '\n';
	delete[] file_name;
	output_file.close();
}
```

```
int work_with_data(AVL_Tree* input_tree, int count_of_elements) {
	cout <<
"----------------------------------------------------------------------\nИтогов
ое считанное дерево:\n\n";
	string dialog_text = "\nВыберите дальнейшее действие:\n1 - вывести
элементы дерева в порядке возрастания в файл и продолжить\n2 - "
		"вывести элементы дерева в порядке возрастания в файл и выйти\n3
- продолжить без вывода в файл\n4 - выйти из программы";
	input_tree->print_tree(&cout);
	cout << "\nПромежуточные операции:\n";
	int* mass_rise_keys = new int[count_of_elements];
	for (int i = 0; i < count_of_elements; i++) {
		mass_rise_keys[i] = input_tree->findmin()->get_key();
//записываем минимальный элемент в массив
		cout << mass_rise_keys[i] << ' ';
		input_tree = input_tree->removemin();  //удаляем его из дерева
	}
	cout << "Элементы в порядке возрастания: ";
	for (int i = 0; i < count_of_elements; i++) {
		cout << mass_rise_keys[i] << ' ';
	}
	cout << "\nНажмите ENTER, чтобы продолжить";
	getchar();
	while (1) {
		switch (input_num(dialog_text)) { //выбор дальнейших действий
пользователем
		case 1: output_tree_elem(mass_rise_keys, count_of_elements);
return 1; break; // вывод элементов в порядке возрастания в файл
		case 2: output_tree_elem(mass_rise_keys, count_of_elements);
return 0; break; // вывод элементов в порядке возрастания в файл и выход
		case 3: return 1; break;  //продолжение работы
		case 4: return 0; break;  //выход
		default: cout << "Команда не распознана!\n"; break;
		}
	}
```

```
	return 0;
}

int console_input() {
	string message = "Введите количество элементов дерева";
	string key_message = "\nВведите значение ключа";
	AVL_Tree* input_tree = nullptr;
	int new_key;
	int count_of_elements = input_num(message);
	if (count_of_elements <= 0) {
		return 1;
	}
	for (int i = 0; i < count_of_elements; i++) {
		new_key = input_num(key_message);
		cout << "Промежуточные операции:\n";
		input_tree = input_tree->insert(new_key); //вставка нового
элемента
		cout << "Текущее состояние дерева:\n";
		input_tree->print_tree(&cout);
	}
	if (work_with_data(input_tree, count_of_elements)) {  //вызов функции
работы с деревом
		return 1;
	}
	return 0;
}

int file_input() {
	int correct_file_name_flag = 0;  //флаг корректного имени файла ввода
	fstream file_input;
	int count_of_elements = 0;
	char* file_name = new char[256];
	while (!correct_file_name_flag) {//цикл до ввода корректного имени
файла
		cout << "Введите имя файла\n\n";
		cin >> file_name;
```

```
            file_input.open(file_name, fstream::in);//открывается файл ввода
            if (file_input.is_open()) {
                  correct_file_name_flag = 1;
            }
            else {
                  cout << "\nФайла с таким именем не найдено!\n";
                  memset(file_name, '\0', 256);
            }
      }
      getchar();//убираем символ переноса строки из потока ввода
      delete[] file_name;
      AVL_Tree* input_tree = nullptr;
      int file_end_flag = 1;//флаг конца файла
      int new_key;
      cout << "Промежуточные операции в ходе создания дерева:\n";
      while (file_end_flag) {
            //считывание элементов дерева из файла
            file_input >> new_key;
            if (file_input.eof()) {//отслеживаем конец файла
                  file_end_flag = 0;
            }
            else {
                  count_of_elements++;
                  input_tree = input_tree->insert(new_key); //вставка нового
элемента
            }
      }
      file_input.close();
      if (work_with_data(input_tree, count_of_elements-1)) {  //вызов
функции работы с деревом
            return 1;
      }
      return 0;
}

int main()
```

```
{
	setlocale(LC_ALL, "rus");
	string start_dialog = "\nВыберите способ ввода данных:\n1 - Ввод с консоли\n2 - Ввод из файла\n3 - Выйти из программы";
	while (1) {
		switch (input_num(start_dialog)) {
		case 1:
			cout << "Выбран ввод с консоли\n\n";
			if (!console_input()) {
				system("pause");
				return 0;
			}
			break;
		case 2:
			cout << "Выбран ввод из файла\n\n";
			if (!file_input()) {
				system("pause");
				return 0;
			}
			break;
		case 3:
			cout << "Выход из программы\n";
			return 0;
			break;
		default:
			cout << "Ответ некорректный!\n\n";
		}
	}
}
```