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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Бинарные деревья.

Студент гр. 9381 _____ Птичкин С. А.

Преподаватель _____ Фирсов М. А.

Санкт-Петербург

2020

Цель работы.

Изучить структуру данных бинарных деревьев и научиться использовать их при решении задач.

Задание.

Вариант 11д.

Формулу вида

<формула> ::= < терминал > | (< формула > < знак > < формула >)

<знак > ::= + | | * < терминал > ::= 0 | 1 | ... | 9 | a | b | ... | z

можно представить в виде бинарного дерева («**дерева-формулы**») с элементами типа Elem=char согласно следующим правилам:

- формула из одного терминала представляется деревом из одной вершины с этим терминалом;

- формула вида (f 1 s f 2) представляется деревом, в котором корень это знак s, а левое и правое поддеревья соответствующие представления формул f 1 и f 2 .

Для всех вариантов (11–17):

- для заданной формулы f построить дерево-формулу t;

- для заданного дерева-формулы t напечатать соответствующую формулу f;

Вариант 11:

- с помощью построения дерева-формулы t преобразовать заданную формулу f из инфиксной формы в префиксную (перечисление узлов t в порядке КЛП) и в постфиксную (перечисление в порядке ЛПК);

- если в дереве-формуле t терминалами являются только цифры, то вычислить (как целое число) значение дерева-формулы t.

Выполнение работы.

Для применения некоторых функций были подключены заголовочные файлы iostream, string, fstream, ctype.h. Исходный код можно посмотреть в Приложении А.

Описание алгоритма.

1) Построение дерева по заданной арифметической формуле использует ЛКП-обход: сначала заполняется левое поддерево(рекурсивным вызовом), а затем заполняется текущий узел, затем правое поддерево(также рекурсивным вызовом). Условием прекращения рекурсии является встреченный терминальный символ или ошибка при вводе.
2) Печать дерева в инфиксной, префиксной и постфиксной записях использует ЛКП, КЛП и ЛПК обходы соответственно. Построение каждой из записей происходит путём слияния строк и отличается лишь в последовательности суммирования строк, полученных при рекурсивных вызовах для правого и левого поддерева и символа арифметической операции. Последовательность суммирования видна из названия каждого обхода.
3) Вычисление значения дерева-формулы использует ЛПК обход. Сначала вычисляется значение для левого поддерева(рекурсивным вызовом), затем для правого(также рекурсивным вызовом), а затем между ними производится операция, взятая из корня.

Структура программы.

Программа разбита на 2 файла: Bin_tree.h, хранящий определение класса и реализацию его методов, и lab3.cpp, в котором реализован пользовательский интерфейс, обработка данных, функции консольного и файлового ввода/вывода.

Пользовательские типы данных.

Структура TREE_AND_REZ.

Предназначена для хранения самого дерева и его числового значения, которое вычисляется только при условии, что все терминалы - цифры.

Класс BIN_TREE.

Класс имитирующий работу бинарного дерева. А именно дерева-формулы. Класс также является шаблонным.

Методы класса BIN_TREE:

1) **BIN_TREE(T data) : data(data)**

   Конструктор с списком инициализации. Принимает на вход данные шаблонного типа и инициализирует ими поле data.

2) **BIN_TREE<T>::BIN_TREE<T>()**

   Конструктор без параметров.

3) **BIN_TREE<T>* BIN_TREE<T>::read_formula(istream* stream, int* file_end_flag, int* enter_flag)**

   Рекурсивный метод считывания дерева из потока. На вход подаётся адрес потока, указатель на флаг конца файла(по умолчанию nullptr) и указатель на флаг считывания конца строки. Сначала метод считывает символ из потока и проверяет его. При считывании конца файла или символа переноса строки метод возвращает нулевой указатель. При считывании

буквы или цифры, динамически создаётся объект бинарного дерева, возвращается указатель на него. При считывании открывающейся скобки, происходит рекурсивный вызов для первого операнда, считывание знака, а затем рекурсивный вызов для второго операнда. При считывании обоих операндов и знака операции метод возвращает созданное дерево-формулу. Также проверяется присутствие закрывающейся скобки. При ошибке считывания любого из операндов или знака, очищается вся выделенная память на всех этапах рекурсивных вызовов.

4) **void BIN_TREE<T>::destroy()**

   Данный метод предназначен для рекурсивной очистки дерева. Метод ничего не принимает. Сначала рекурсивно удаляются поддеревья текущего узла, а затем и сам узел. Метод ничего не возвращает.

5) **void BIN_TREE<T>::set_value(T value)**

   Метод предназначен для установки конкретного значения в узел бинарного дерева. На вход подаётся само значение. Функция ничего не возвращает.

6) **bool BIN_TREE<T>::is_terminals_digit()**

   Метод предназначен для проверки терминалов всего бинарного дерева на принадлежность цифрам. Метод ничего не принимает на вход. Идёт рекурсивный обход дерева, встретив букву метод возвращает ложь, если цифру - true.

7) **int BIN_TREE<T>::rec_calculate()**

   Данный метод запускается только в случае, когда все терминалы в дереве - цифры. Метод ничего не принимает на вход. Метод рекурсивно обходит дерево, и встретив цифру возвращает её целочисленное значение. Встретив знак операции, происходит рекурсивный вызов для левого и

правого поддерева. Результаты проходят операцию, которая указана в знаке. Её результат возвращается методом. При этом выводятся промежуточные операции.

8) **string BIN_TREE<T>::infix_form()**

Данный метод предназначен для печати инфиксной формы дерева-формулы. На вход метод ничего не принимает. Дерево рекурсивно обходится и на каждом этапе происходит суммирование символов из левого поддерева, корня и правого поддерева(ЛКП обход). Суммирование происходит в строке, которую и возвращает метод.

9) **string BIN_TREE<T>::postix_form()**

Данный метод предназначен для печати инфиксной формы дерева-формулы. На вход метод ничего не принимает. Дерево рекурсивно обходится и на каждом этапе происходит суммирование символов из левого поддерева, правого поддерева и корня(ЛПК обход). Суммирование происходит в строке, которую и возвращает метод.

10) **string BIN_TREE<T>::prefix_form()**

Данный метод предназначен для печати инфиксной формы дерева-формулы. На вход метод ничего не принимает. Дерево рекурсивно обходится и на каждом этапе происходит суммирование символов из корня, левого поддерева и правого поддерева(КЛП обход). Суммирование происходит в строке, которую и возвращает метод.

11) **bool BIN_TREE<T>::isLeaf()**

Метод предназначен для определения, является ли элемент дерева листом. Метод ничего не принимает и возвращает false, если поддеревья не пусты, иначе - true.

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. Далее вызывается рекурсивная функция считывания дерева из файла. Пока не достигнут конец файла действие повторяется. Затем файл закрывается и вызывается data_analis, функция работы с деревьями. Функция возвращает то же значение, что и data_analis.

3) Функция console_input. int console_input()

Функция не принимает никаких параметров. В начале объявляются все необходимые для работы переменные. Затем пользователь вводит количество деревьев. После чего вызывается рекурсивная функция считывания дерева из консоли. Затем вызывается data_analis, функция работы с деревьями. Функция возвращает то же значение, что и data_analis.

4)int data_analis(TREE_AND_REZ* data_mass, int count_of_tree)

Функция принимает на вход массив структур TREE_AND_REZ и

количество элементов в нём. Сначала деревья выводится на экран в инфиксной форме. Затем каждое дерево последовательно выводится в инциксной, префиксной и постфиксной форме, с использованием соответствующих методов. После проверяется все ли терминалы дерева - цифры, если да, то выделяется память под число, адрес памяти записывается в поле rez структуры TREE_AND_REZ, затем вызывается метод подсчёта значения дерева, результат записывается в rez и выводится на экран. Пройдя все деревья в цикле, пользователь попадает в меню, где выбирает сохранить ли полученные результаты в файл, продолжить или завершить программу. При записи вызывается функция data_save. Затем память очищается функцией clear_memory. Функция возвращает 1 для продолжения работы, 0 - для завершения.

5) void data_save(TREE_AND_REZ* data_mass, int count_of_tree)

Функция принимает на вход массив структур TREE_AND_REZ и количество элементов в нём. Сначала пользователю предлагается назвать имя файла для записи. Затем файл открывается и в него последовательно записываются деревья в инфиксной, постфиксной и префиксной записи, а также их значение, если все терминалы - цифры. Файл закрывается и функция ничего не возвращает.

6) Функция void clear_memory(TREE_AND_REZ* data_mass, int count_of_tree)

Данная функция принимает на вход массив структур TREE_AND_REZ и количество элементов в нём. Затем в цикле последовательно вызываются методы уничтожения деревьев, а также очищается память под указатель на число - значение дерева, если такое имеется. В конце очищается память под сам массив структур, функция ничего не возвращает.

7) 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.

Результаты теста входных данных представлены в таблице 1.

Таблица 1- Результаты тестирования

№ Теста	Входные данные	Выходные данные
1	(4-3)	Формула: (4-3) Префиксная форма: -43 Постфиксная форма: 43- Значение: 1
2	(	Пустое дерево
3	(d+	Пустое дерево
4	(d+s	Пустое дерево
5	(2+3)wdqwdq	Формула: (2+3) Префиксная форма: +23 Постфиксная форма: 23+ Значение: 5

6	(3+(3*0))	Формула: (3+(3*0)) Префиксная форма: +3*30 Постфиксная форма: 330*+ Значение: 3
7	((e+d)-(3*2))	Формула: ((e+d)-(3*2)) Префиксная форма: -+ed*32 Постфиксная форма: ed+32*-
8	(((2+3)-(4-3))*((8*0)+(1*5)))	Формула: (((2+3)-(4-3))*((8*0)+(1*5))) Префиксная форма: *-+23-43+*80*15 Постфиксная форма: 23+43--80*15*+* Значение: 20

Вывод.

Были изучены бинарные деревья, различные методы их обхода и работа с ними.

Приложение А

Исходный код программы

Название файла: Bin_tree.h

```
#pragma once
#include <iostream>
#include <fstream>
#include <ctype.h>
#include <string>
using namespace std;

template<typename T>
class BIN_TREE {
private:
	T data;
	BIN_TREE* left = nullptr;
	BIN_TREE* right = nullptr;
public:
	BIN_TREE(T data) : data(data) {}
	BIN_TREE();
	void destroy();
	BIN_TREE* read_formula(istream* stream, int* file_end_flag = nullptr,
int* error_flag = nullptr);
	string infix_form();
	string prefix_form();
	string postfix_form();
	int rec_calculate();
	bool is_terminals_digit();
	void set_value(T value);
	bool isLeaf();
};

template<typename T>
BIN_TREE<T>::BIN_TREE<T>() {}

template<typename T>
```

```
BIN_TREE<T>* BIN_TREE<T>::read_formula(istream* stream, int* file_end_flag,
int* error_flag = nullptr) { //рекурсивная функция считывания дерева-формулы
	T simvol;
	BIN_TREE* q;
	simvol = stream->get();
	if (simvol == EOF) {
		*file_end_flag = 0;
		return nullptr;
	}
	if (simvol == '\n') {
		if(error_flag)
		*error_flag = 1;
		return nullptr;
	}
	if (simvol == '(') {
		q = new BIN_TREE<T>;
		q->left = read_formula(stream, file_end_flag, error_flag);
//рекурсивный вызов для левого терминала
		if (q->left == nullptr) {
			delete q;
			return nullptr;
		}
		q->data = stream->get(); //считывание знака
		if (strchr("+-*", q->data) == nullptr) {
			if (simvol == '\n') {
				if (error_flag)
				*error_flag = 1;
			}
			q->destroy();
			return nullptr;
		}
		q->right = read_formula(stream, file_end_flag, error_flag);
//рекурсивный вызов для правого терминала
		if (q->right == nullptr) {
			q->destroy();
			return nullptr;
```

```
            }
            simvol = stream->get();
            if (simvol != ')') { //проверка на закрытие скобок
                  if (simvol == '\n') {
                        if (error_flag)
                              *error_flag = 1;
                  }
                  q->destroy();
                  return nullptr;
            }
            return q;
      }
      if ((isalpha(simvol)) || (isdigit(simvol))) {
            return new BIN_TREE<T>(simvol);
      }
      else {
            return nullptr;
      }
}

template<typename T>
void BIN_TREE<T>::destroy() {//рекурсивная функция очистки дерева
      if (this != nullptr) {
            left->destroy();
            right->destroy();
            delete this;
      }
}

template<typename T>
void BIN_TREE<T>::set_value(T value) { //функция установки значения корня
      if (this != nullptr)
            data = value;
}

template<typename T>
```

```
bool BIN_TREE<T>::is_terminals_digit() { //проверка, что все терминалы
дерева - цифры
	if (this == nullptr) {
		return false;
	}
	else if (strchr("+-*", data)){
		return left->is_terminals_digit() && right->is_terminals_digit();
//рекурсивный вызов для правого и левого терминала
	}
	else if (isdigit(data)) {
		return true;
	}
	else {
		return false;
	}
}

template<typename T>
int BIN_TREE<T>::rec_calculate() { //функция подсчёта значения дерева
	if (isdigit(data)) { //если значение корня - цифру, возвращаем её
		return int(data)-48;
	}
	int left_num;
	int right_num;
	int rez;
	switch (data)
	{
	case '+':
		left_num = left->rec_calculate();
		right_num = right->rec_calculate();
		rez = left_num + right_num; //вычисление значения формулы
		cout << left_num << " + " << right_num << " = " << rez<<'\n';
//промежуточные данные
		return rez; break;
	case '-':
		left_num = left->rec_calculate();
```

```
            right_num = right->rec_calculate();
            rez = left_num - right_num;   //вычисление значения формулы
            cout << left_num << " - " << right_num << " = " << rez <<
'\n';//промежуточные данные
            return rez; break;
      case '*':
            left_num = left->rec_calculate();
            right_num = right->rec_calculate();
            rez = left_num * right_num;   //вычисление значения формулы
            cout << left_num << " * " << right_num << " = " << rez <<
'\n';//промежуточные данные
            return rez; break;
      }
}

template<typename T>
string BIN_TREE<T>::infix_form() {
      if (this == nullptr)
            return "";
      // преобразование в инфиксную (ЛКП) запись
      string str;
      if (!isLeaf()) str += '(';
      if (left)
            str += left->infix_form(); // добавляется левое поддерево
      str += data; // добавляется корень
      if (right)
            str += right->infix_form(); // добавляется правое поддерево
      if (!isLeaf()) str += ')';
      return str;
}

template<typename T>
string BIN_TREE<T>::postfix_form(){
      if (this == nullptr)
            return "";
      // преобразование в постфиксную (ЛПК) запись
```

```
	string str;
	if (left)
		str += left->postfix_form(); // добавляется левое поддерево
	if (right)
		str += right->postfix_form(); // добавляется правое поддерево
	str += data; // добавляется корень
	return str;
}

template<typename T>
string BIN_TREE<T>::prefix_form() {
	if (this == nullptr)
		return "";
	// преобразование в префиксную (КЛП) запись
	string str;
	str += data; // добавляется корень
	if (left)
		str += left->prefix_form(); // добавляется левое поддерево
	if (right)
		str += right->prefix_form(); // добавляется правое поддерево
	return str;
}

template<typename T>
bool BIN_TREE<T>::isLeaf(){ //проверка на лист
	return !left && !right;
}
```

Файл lab3.cpp:

```
#include "stdafx.h"
#include "BIN_tree.h"
#include <iostream>

struct tree_and_rez {
	BIN_TREE<char>* tree;
	int* rez = nullptr;
};
```

```
typedef struct tree_and_rez TREE_AND_REZ;

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 data_save(TREE_AND_REZ* data_mass, int count_of_tree) {//сохранение данных в файл
	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;
	output_file.open(file_name, fstream::out | fstream::app);//открытие или создание файла на запись
	for (int i = 0; i<count_of_tree; i++) {
		if (data_mass[i].tree == nullptr) {
			output_file << "Пустое дерево" << '\n';
		}
		else { //запись результатов в файл
			output_file << "Формула: " << data_mass[i].tree->infix_form() << '\n';
			output_file << "Префиксная форма: " << data_mass[i].tree->prefix_form() << '\n';
			output_file << "Постфиксная форма: " << data_mass[i].tree->postfix_form() << '\n';
```

```
            }
            if(data_mass[i].rez!=nullptr)
                  output_file << "Значение: " << *data_mass[i].rez << '\n';
            output_file << '\n';
      }
      delete[] file_name;
      output_file.close();
}

void clear_memory(TREE_AND_REZ* data_mass, int count_of_tree) {//функция
очистки памяти
      for (int i = 0; i<count_of_tree; i++) {
            if (data_mass[i].tree != nullptr) {
                  data_mass[i].tree->destroy();  //вызов рекурсивной очистки
дерева
            }
            if (data_mass[i].rez != nullptr) {
                  delete data_mass[i].rez; //очищаем значение дерева, если
оно посчитано
            }
      }
      delete[] data_mass;
}

int data_analis(TREE_AND_REZ* data_mass, int count_of_tree) {
      cout <<
"------------------------------------------------------------------------
---------------\nИсходные данные:\n";
      string dialog_text = "\nВыберите дальнейшее действие:\n1 - сохранить
данные в файл и продолжить\n2 - сохранить данные в файл и выйти\n3 -
продолжить без сохранения\n4 - выйти без сохранения";
      for (int i = 0; i < count_of_tree; i++) {
            cout << "Дерево-формула " << i + 1 << ": ";
            if (data_mass[i].tree == nullptr) {
                  cout << "Пустое дерево" << '\n';
            }
```

```
            else {
                  cout << data_mass[i].tree->infix_form() << '\n';
//исходный вид формулы
            }
      }
      cout << "Обработка данных:\n\n";
      for (int i = 0; i < count_of_tree; i++) {
            cout << "Дерево-формула " << i + 1 << ":\n";
            if (data_mass[i].tree == nullptr) {
                  cout << "Пустое дерево" << "\n\n";
            }
            else {
                  cout << "Инфиксная форма: " <<
data_mass[i].tree->infix_form() << '\n';
                  cout << "Префиксная форма: " <<
data_mass[i].tree->prefix_form() << '\n';
                  cout << "Постфиксная форма: " <<
data_mass[i].tree->postfix_form() << '\n';
                  if (data_mass[i].tree->is_terminals_digit()) {
                        cout << "Промежуточные данные:\n";
                        data_mass[i].rez = new int;
                        *data_mass[i].rez =
data_mass[i].tree->rec_calculate(); //запись значения дерева-формулы
                        cout << "Итоговое значение дерева: " <<
*data_mass[i].rez << "\n";
                  }
                  cout << '\n';
            }
      }
      cout << "Нажмите ENTER, чтобы продолжить";
      getchar();
      while (1) {
            switch (input_num(dialog_text)) { //выбор дальнейших действий
пользователем
```

```
            case 1: data_save(data_mass, count_of_tree);
clear_memory(data_mass, count_of_tree); return 1; break;//сохранение и
очистка данных
            case 2: data_save(data_mass, count_of_tree);
clear_memory(data_mass, count_of_tree); return 0; break;
            case 3: clear_memory(data_mass, count_of_tree); return 1; break;
            case 4: clear_memory(data_mass, count_of_tree); return 0; break;
            default: cout << "Команда не распознана!\n"; break;
            }
      }
}

int console_input() {
      string message = "Введите количество деревьев";
      int count_of_tree = input_num(message);
      int error_flag = 0;
      if (count_of_tree <= 0) {
            return 1;
      }
      TREE_AND_REZ* data_mass = new TREE_AND_REZ[count_of_tree];
      for (int i = 0; i < count_of_tree; i++) {
            cout << "Дерево-формула " << i + 1 << ": ";
            data_mass[i].tree = data_mass[i].tree->read_formula(&cin,
nullptr, &error_flag); //считывание дерева из консоли
            if (error_flag == 0) {
                  if (data_mass[i].tree != nullptr) {//считываем оставшиеся
символы
                        while (getchar() != '\n');
                  }
            }
            error_flag = 0;
            /*if (data_mass[i].tree != nullptr) {
                  while (getchar() != '\n');
            }*/
      }
```

```
	if (data_analis(data_mass, count_of_tree)) {//вызов функции анализа
данных
		return 1;
	}
	return 0;
}

int file_input() {
	int count_of_tree = 0;
	int c;
	int correct_file_name_flag = 0;  //флаг корректного имени файла ввода
	fstream file_input;
	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;
	int buff = 10;//буффер количества строк
	int file_end_flag = 1;//флаг конца файла
	int error_flag = 0;
	TREE_AND_REZ* data_mass = new TREE_AND_REZ[buff];
	TREE_AND_REZ* rezerv_data_mass;
	while (file_end_flag) {
		if (count_of_tree == buff) {//проверка на заполнение буффера
			buff += 10;
```

```
                rezerv_data_mass = new TREE_AND_REZ[buff];
                for (int i = 0; i<buff - 10; i++) {
                      rezerv_data_mass[i].tree = data_mass[i].tree;
                }
                delete[] data_mass;
                data_mass = rezerv_data_mass;
                rezerv_data_mass = nullptr;
          }
          //считывание дерева из файла
          data_mass[count_of_tree].tree =
data_mass[count_of_tree].tree->read_formula(&file_input, &file_end_flag,
&error_flag);
          if (error_flag == 0) {
                if ((file_end_flag) && (data_mass[count_of_tree].tree !=
nullptr)) {//считываем оставшиеся символы
                      c = ((istream*)(&file_input))->get();
                      while ((c != '\n') && (c != EOF)) {
                            c = ((istream*)(&file_input))->get();
                      }
                }
          }
          error_flag = 0;
          count_of_tree++;
     }
     file_input.close();
     if (data_analis(data_mass, count_of_tree - 1)) {//вызов функции
анализа данных
          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";
			system("pause");
			return 0;
			break;
		default:
			cout << "Ответ некорректный!\n\n";
		}
	}
}
```