МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
ДОНЕЦКОЙ НАРОДНОЙ РЕСПУБЛИКИ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ
по дисциплине
«Объектно-ориентированное программирование»
Часть 2

Донецк
2022

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
ДОНЕЦКОЙ НАРОДНОЙ РЕСПУБЛИКИ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА «КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ И ДИЗАЙН»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ
по дисциплине
«Объектно-ориентированное программирование»
Часть 2

для обучающихся по направлениям подготовки
02.03.01 «Математика и компьютерные науки»,
09.03.02 «Информационные системы и технологии»
всех форм обучения

РАССМОТРЕНО
на заседании кафедры
компьютерного моделирования и
дизайна
Протокол № 4 от 29.12.2021 г.

УТВЕРЖДЕНО
на заседании учебно-издательского
совета ДОННТУ
Протокол № 2 от 16.02.2022 г.

Донецк
2022

УДК 004.43:004.8(076)
М54

Составитель:
Павлий Виталий Александрович – кандидат технических наук, доцент кафедры компьютерного моделирования и дизайна ГОУВПО «ДОННТУ»

М54 **Методические указания к выполнению лабораторных работ по дисциплине «Объектно-ориентированное программирование» Ч. 2** : для обучающихся по направлениям подготовки 02.03.01 «Математика и компьютерные науки», 09.03.02 «Информационные системы и технологии» всех форм обучения / ГОУВПО «ДОННТУ», Каф. компьютерного моделирования и дизайна ; сост. В. А. Павлий. – Донецк : ДОННТУ, 2022. – Систем. требования: Acrobat Reader. – Загл. с титул. экрана.

Методические указания разработаны с целью оказания помощи обучающимся в получении практических навыков программирования на языке C++ и включают теоретический материал, задания, контрольные вопросы, а также примеры выполнения лабораторных работ по основным темам, изучаемым во второй части курса. Для каждой лабораторной работы предусмотрено по 50 индивидуальных вариантов заданий.

УДК 004.43:004.8(076)

Содержание

Введение

Методические указания к выполнению лабораторных работ по дисциплине «Объектно-ориентированное программирование» (часть 2) разработаны для оказания помощи обучающимся в получении практических навыков программирования на языке C++ и обобщения теоретического материала по второй части курса.

Методические указания включают 3 лабораторные работы по основным темам второй части курса. Каждая лабораторная работа включает основные теоретические сведения, необходимые для выполнения лабораторной работы, постановку задания к лабораторной работе, пример выполнения работы в виде программного кода на языке C++. После выполнения лабораторной работы обучающиеся оформляют отчет, содержание которого также включено в состав лабораторной работы. Для контроля и закрепления теоретического материала в каждой лабораторной работе имеются контрольные вопросы по изучаемой теме.

Каждая лабораторная работа включает 50 индивидуальных вариантов заданий равнозначной степени сложности.

Лабораторная работа №6
Работа с исключениями в C++

Цель работы: закрепить теоретические знания использования исключений для перехвата и обработки ошибок, возникающих в процессе работы программы, изучить создание собственных классов исключений на основе базового класса std::Exception, а также возбуждение собственных исключений.

6.1. Основные теоретические сведения

Исключительная ситуация, или *исключение* – это возникновение непредвиденного или аварийного события, которое может порождаться некорректным использованием программы. Например, это деление на ноль или обращение по несуществующему адресу памяти. Обычно эти события приводят к завершению программы с системным сообщением об ошибке. C++ дает программисту возможность восстанавливать программу и продолжать ее выполнение. Механизм исключений предназначен только для событий, которые происходят в результате работы самой программы и указываются явным образом. Исключения возникают тогда, когда некоторая часть программы не смогла сделать то, что от нее требовалось. При этом другая часть программы может попытаться сделать что-нибудь иное.

Исключения позволяют логически разделить вычислительный процесс на две части – обнаружение аварийной ситуации и ее обработка. Это важно не только для лучшей структуризации программы. Главной причиной является то, что функция, обнаружившая ошибку, может не знать, что предпринимать для ее исправления, а использующий эту функцию код может знать, что делать, но не уметь определить место возникновения. Это особенно актуально при использовании библиотечных функций и программ, состоящих из многих модулей. Другое достоинство исключений состоит в том, что для передачи информации об ошибке в вызывающую функцию не требуется применять возвращаемое значение, параметры или глобальные переменные, поэтому интерфейс функций не раздувается. Это особенно важно, например, для конструкторов, которые по синтаксису не могут возвращать значение.

Известно, что при вызове каждой функции в стеке создается область памяти для хранения локальных переменных и адреса возврата в вызывающую функцию.

Термин *стек вызовов* обозначает последовательность вызванных, но еще не завершившихся функций. *Раскручиванием стека* называется процесс освобождения памяти из-под локальных переменных и возврата управления вызывающей функции. Когда функция завершается, происходит естественное раскручивание стека. Тот же самый механизм используется и при обработке исключений. Поэтому после того, как исключение было зафиксировано, исполнение не может быть продолжено с точки генерации исключения. Подробнее этот механизм рассматривается ниже.

Синтаксис исключений

Ключевое слово try служит для обозначения *контролируемого блока* — кода, в котором может генерироваться исключение. Блок заключается в фигурные скобки:

```
try
{
   ...
}
```

Все функции, прямо или косвенно вызываемые из try-блока, также считаются ему принадлежащими. *Генерация (порождение) исключения* происходит по ключевому слову throw, которое употребляется либо с параметром, либо без него:

```
throw [ выражение ];
```

Тип выражения, стоящего после throw, определяет тип порождаемого исключения. При генерации исключения выполнение текущего блока прекращается, и происходит поиск соответствующего обработчика и передача ему управления. Как правило, исключение генерируется не непосредственно в try-блоке, а в функциях, прямо или косвенно в него вложенных. Не всегда исключение, возникшее во внутреннем блоке, может быть сразу правильно обработано. В этом случае используются вложенные контролируемые блоки, и исключение передается на более высокий уровень с помощью ключевого слова throw без параметров.

Обработчики исключений начинаются с ключевого слова *catch*, за которым в скобках следует тип обрабатываемого исключения. Они должны располагаться непосредственно за try-блоком. Можно записать один или несколько обработчиков в соответствии с типами обрабатываемых исключений. Синтаксис обработчиков напоминает определение функции с одним параметром — типом исключения. Существует три формы записи:

```
catch(Type имя){/* тело обработчика */}
catch(Type){/* тело обработчика */}
catch(...){/* тело обработчика */}
```

Первая форма применяется, когда имя параметра используется в теле обработчика для выполнения каких-либо действий – например, вывода информации об исключении. Вторая форма не предполагает использования информации об исключении, играет роль только его тип. Многоточие вместо параметра обозначает, что обработчик перехватывает все исключения. Так как обработчики просматриваются в том порядке, в котором они записаны, обработчик третьего типа следует помещать после всех остальных.

В качестве типа обрабатываемого исключения в языке C++ допускается использовать любой тип данных – как примитивный, так и сложный объект. Например, примитивный тип int может представлять код ошибки, а примитивный тип char* – сообщение об ошибке. Однако, если возникшая ошибка должна быть описано более детально или при возникновении ошибки должны выполняться некоторые действия (например, автоматическая отправка Email-сообщения разработчику программы) примитивным типом уже не обойтись. В этом случае программист создает специальный класс. Поля этого класса несут детальную информацию о возникшей ошибке (например, код

ошибки, тип ошибки, сообщение об ошибке, номер строки, в которой произошла ошибка, имя файла, в котором произошла ошибка и т.п.), а методы этого класса осуществляют действия по ее устранению (отправка Email разработчику, запись сообщения об ошибке в лог-файл и т.п.). Подобный класс позволяет более детально описать возникшую ошибку и предпринять действия для ее устранения. Обычно такой класс наследуют от стандартного класса std::Exception.

Ранее было указано, что для представления ошибки может быть использован примитивный тип данных, например int. В этом случае код генерации и обработки ошибки может выглядеть например, так:

```
try
{
   ***
   throw -1; // Возникла ошибка с кодом -1. «-1» это условный код
             //ошибки, который определяет программист, и поэтому
             // только он знает, что этот код ошибки означает
}
catch (int error) {
   cout << "Возникла ошибка с кодом: " <<error;
}
```

В случае использования класса, описывающего возникшую ошибку, код генерации и обработки ошибки будет выглядеть следующим образом:

```
try
{
   ***
   throw new MyException;
}
catch (MyException error) {
   cout << "Возникла ошибка с кодом: " << error -> getCode();
   cout << " в строке: " << error -> getLine();
   error -> sendEmail();
}
```

Как видно из приведенного кода, в этом случае можно вывести более детальную информацию о возникшей ошибке на экран, а также выполнить дополнительные действия, например, отправить Email-сообщение разработчику. Разумеется, метод отправки сообщения должен быть реализован в классе MyException. Поскольку данный способ дает более широкие возможности контроля обработки возникшей ошибки, заметим, что в современных языках программирования *использование примитивных типов для представления ошибок запрещено*.

После обработки исключения управление передается первому оператору, находящемуся непосредственно за обработчиками исключений. Туда же, минуя код всех обработчиков, передается управление, если исключение в try-блоке не было сгенерировано.

6.2. Задание к лабораторной работе

Задание к лабораторной работе состоит из двух частей, *никак не связанных между собой.*

В первой части необходимо реализовать приложение на языке C++, которое вычисляет выражение в соответствии с вариантом задания,

представленном в таблице 6.1. **При этом программа должна корректно обрабатывать все возникшие ошибки** (ввода исходных данных – например, пользователь может ввести вместо числа текст, расчета значения выражения – например, может возникнуть деление на 0 и т.п.) при помощи исключений, использующих примитивный тип данных. Для этого выражение необходимо тщательно проанализировать на предмет возможных ошибок.

Во второй части работы первым **делом необходимо разработать класс-наследник от стандартного класса std::Exception**, детально описывающий ошибку или множество ошибок, которые могут возникнуть при возникновении ситуации, представленной в таблице 6.2. Этот класс также **должен содержать функцию записи в файл детальной информации о возникшей ошибке**. Далее необходимо реализовать приложение, создающее исключения, в которых ошибка представлена этим классом.

В обеих частях работы необходимо продемонстрировать на скриншотах возникновение всех возможных ошибок и их обработку. Для этого следует самостоятельно подобрать параметры условия таким образом, чтобы ошибка проявилась.

6.3. Пример выполнения лабораторной работы

Часть 1. Расчет выражения $y = \dfrac{\dfrac{a+b}{c} - tg(b)}{a}$.

Файл lab6_1.cpp

```
/*  Расчет выражения: y = ((a+b)/c - tg(b))/a */
#define _USE_MATH_DEFINES
#define EPS 0.0000001
#include <iostream>
#include <cmath>
using namespace std;
// Функция вводит значение типа double,
// если это невозможно - кидает исключение
double getValue()
{
    double a;
    cin >> a;
    // Проверка на корректность ввода
    if (cin.fail())
        throw 1;
    while (cin.get() != '\n')
        throw 1;
    return a;
}

void main()
{
    setlocale(0, "RUS");
    try
    {
        // Ввод исходных данных
        cout << "Введите a: ";
```

```
        double a = getValue();
        cout << "Введите b: ";
        double b = getValue();
        cout << "Введите c: ";
        double c = getValue();
        // Вычисление (a+b)/c
        if (abs(c) < EPS)
            throw 2;
        double res0 = (a + b) / c;
        // Вычисление tg(b)
        // Сначала нужно нормализовать b по формулам
        // приведения tg(PI + a) = tg(a)
        if (b < 0)
            b += floor(b / M_PI) * M_PI;
        else
            b -= floor(b / M_PI) * M_PI;
        if (b < 0) b += M_PI;
        if (abs(b - M_PI_2) < EPS)
            throw 3;
        double res1 = tan(b);
        // Окончательное вычисление выражения
        if (abs(a) < EPS)
            throw 4;
        double y = (res0 - res1) / a;
        // Вывод результата
        cout << "Результат: y = " << y << endl;
    }
    catch (int i)
    {
        switch (i)
        {
            case 1:
                cout << "Введено некорректное значение";
                break;
            case 2:
                cout << "Ошибка деления на 0 при вычислении выражения (a+b)/c";
                break;
            case 3:
                cout << "Ошибка вычисления tg(b)";
                break;
            case 4:
                cout << "Ошибка деления на 0 при вычислении всего выражения";
                break;
        }
    }
    catch (...)
    {
        cout << "Неизвестная ошибка";
    }
}
```

Часть 2. В массиве дробных чисел заданы только целые числа.

Файл MyException.h

```
#include <exception>
#include <fstream>
#include <iostream>
using namespace std;
class MyException : public exception
{
    private:
```

```
        // Файл, в котором произошла ошибка
        char* file;
        // Строка, в которой произошла ошибка
        int line;
        // Код ошибки
        int code;
        // Сообщение об ошибке
        char* message;
    public:
        // Конструктор
        MyException(char* file, int line, int code, char* message);
        // Добавляет сообщение об ошибке в лог-файл
        void addToLog();
        // Выводит сообщение о возникшей ошибке на экран
        void displayMessage();
};
```

Файл MyException.cpp

```
#include "MyException.h"
MyException::MyException(char* file, int line, int code, char* message)
{
    this -> file = file;
    this -> line = line;
    this -> code = code;
    this -> message = message;
}
void MyException::addToLog()
{
    ofstream logFile;
    logFile.open("log.txt", ios::out | ios::app);
    if (logFile)
    {
        logFile << "Дата: " << __DATE__ << ", время: " << __TIME__;
        logFile << ", файл: " << this -> file << ", строка: " << this -> line;
        logFile << ", код ошибки: " << this -> code << ", текст ошибки: ";
        logFile << this -> message << endl;
        logFile.close();
    }
    else
        cout << "Не удалось открыть файл лога для записи" << endl;
}
void MyException::displayMessage()
{
    cout << "Возникла ошибка с кодом: " << this -> code;
    cout << ", текст ошибки: " << this -> message;
}
```

Файл lab6_2.cpp

```
#include "MyException.h"
#define eps 0.000001
using namespace std;
// Функция вводит значение типа double,
// если это невозможно - кидает исключение
double getValue()
{
    double a;
    cin >> a;
    // Проверка на корректность ввода
    if (cin.fail())
        throw new MyException(__FILE__, __LINE__, 2, "Введено некорректное значение");
```

```
    while (cin.get() != '\n')
        throw new MyException(__FILE__, __LINE__, 2, "Введено некорректное значение");
    return a;
}
// Главная функция
void main()
{
    // Массив дробных чисел
    double *data;
    int count;
    // Настройка локали (для корректного отображения русских букв)
    setlocale(0, "RUS");
    try
    {
        cout << "Введите количество элементов массива: ";
        count = (int)getValue();
        if (count < 1)
           throw new MyException(__FILE__, __LINE__, 1,
               "Количество элементов массива должно быть положительным числом");
        data = new double[count];
        for (int i = 0; i < count; i++)
        {
            cout << "Введите элемент №" << (i + 1) << ": ";
            data[i] = getValue();
        }
        // Счетчик целых элементов
        int countInt = 0;
        for (int i = 0; i < count; i++)
        {
            if (abs(data[i] - floor(data[i]) < eps))
                countInt++;
        }
        // Все элементы целые - сгенерировать исключение
        if (countInt == count)
            throw new MyException(__FILE__, __LINE__, 3,
               "Все элементы введенного массива являются целыми числами");
        // Вывести сообщение
        cout << "Массив успешно прошел проверку на целые числа";
        // Почистить память
        delete data;
    }
    catch (MyException *e)
    {
        // Добавляем сообщение о возникщей ошибке в лог-файл
        e -> addToLog();
        // Выводим сообщение на экран
        e -> displayMessage();
    }
    catch (...)
    {
        cout << "Неизвестная ошибка";
    }
}
```

6.4. Содержание отчета

1. Титульный лист.
2. Условие лабораторной работы.
3. Текст программы.
4. Экранные формы с примерами работы программы.

6.5. Контрольные вопросы

1. Что такое исключительная ситуация?
2. Как происходит обработка ошибок в языке C++?
3. Сколько блоков catch может существовать в конструкции try-catch?
4. Что такое стек вызовов?
5. Каким образом генерируются исключения?

6.6. Варианты заданий

В таблицах 6.1. и 6.2 приведены индивидуальные варианты заданий для работы с исключениями для выполнения первой и второй части работы.

Задания из таблиц 6.1 и 6.2 никак не взаимосвязаны друг с другом, вся лабораторная работа состоит из двух отдельных независимых частей.

Таблица 6.1.

Индивидуальные варианты заданий для выполнения первой части работы

№ варианта	Задание
1	Расчет выражения: $u = \sqrt{\frac{a+b}{c-d}} + b - a\frac{d}{b}$
2	Расчет выражения: $u = \sqrt{\frac{b-a}{d}} + \ln(b) + c\frac{a}{b}$
3	Расчет выражения: $u = \sqrt{\frac{b}{2} + \frac{1}{x} \cdot \frac{bx}{b-x^2}}$
4	Расчет выражения: $u = (d-a)/b - \sqrt{\frac{b}{d} + \log(c)}$
5	Расчет выражения: $u = \sqrt{\frac{c}{5}} \cdot \frac{yz}{x-x^2} - \frac{1}{2}c$
6	Расчет выражения: $u = \frac{\sin 2x \cdot \cos 4x}{y} + \sqrt{\frac{1}{x}}$
7	Расчет выражения: $u = \sqrt{\frac{3a-4b}{x} - \lvert ax+bx \rvert}$
8	Расчет выражения: $u = \frac{\sqrt{\frac{a^2+b^2}{c+d}}}{a}\cos(ab-cd)$
9	Расчет выражения: $u = \frac{yx+c}{b} + \frac{a}{b^2}\ln(yx+c)$
10	Расчет выражения: $u = \frac{x}{y}\sin c - \left(\frac{x^2}{y} - \frac{1}{x^3}\right)\cos d$
11	Расчет выражения: $u = \frac{a+5b}{\sqrt{x-3b}}\log_c \frac{a}{x} + \cos(a-8b)tg(x)$

12	Расчет выражения: $u = \dfrac{b-4a}{\cos\left(\dfrac{4b-c}{\sqrt{a-2}}\right)+tg\left(b^2-ac\right)}$
13	Расчет выражения: $u = \dfrac{x}{y}\sin c - \left(\dfrac{x^2}{y} - \dfrac{1}{\cos x + tgy}\right) - c$
14	Расчет выражения: $u = \dfrac{x^3 + 3x^{2+\sqrt{a}}}{c+5}\ln\left(x+3y+\sqrt{a+c}\right)$
15	Расчет выражения: $u = \cos\left(\dfrac{a-4}{\sqrt{b+a-2c}}\right)\dfrac{\sqrt{tg\left(\dfrac{5}{4ab}\right)}}{c-a}$
16	Расчет выражения: $u = \dfrac{x^2-2xy}{3y} - \dfrac{\sqrt{y}}{x^2-y}tg\left(\dfrac{x+5y}{8-y^3}\right)$
17	Расчет выражения: $u = \dfrac{a+4b\dfrac{c}{a-b}}{c\left(\dfrac{\dfrac{1}{ax+b}+\dfrac{y}{c}\ln(a+x)}{b+x}\right)}$
18	Расчет выражения: $u = \dfrac{2b}{x^2-\sqrt{b}} - \dfrac{1}{x-b}\cos(2a-b)$
19	Расчет выражения: $u = \sqrt{a - \dfrac{4b\cos(a-2c)}{\sin\left(\dfrac{2b-4c}{a+ctg(b)}\right)}}$
20	Расчет выражения: $u = \dfrac{\sin\left(ba-\sqrt{b+4c}\right)}{\dfrac{a+b\cos(b-4c)}{ab} - ctg(ba-3)}$
21	Расчет выражения: $u = \dfrac{c-1}{c}tg\left(\dfrac{d}{2c-a}\right)\sqrt{\dfrac{ab\cdot\cos(c+d)}{b-4\sin(c-d)}}$
22	Расчет выражения: $u = \dfrac{\dfrac{a+b}{a-b}\ln\left(a^2+\sqrt{\dfrac{ab+4}{\sin(b-a)}}+b^2\right)}{x+y}$
23	Расчет выражения: $u = \dfrac{\dfrac{a+b^2}{c}\ln\left(c+\sqrt{2adb}+d^2\right)}{a+b}$
24	Расчет выражения: $u = \dfrac{2a}{4a^2-b^2} + \dfrac{1}{b-2a}$
25	Расчет выражения: $u = \sqrt{\dfrac{a-\sin(2b-c)}{b+\cos(3a+c)}}\cdot\dfrac{a+b}{4c}$
26	Расчет выражения: $u = \dfrac{8a}{4a^2+x} + \dfrac{3}{b-\sqrt{2a}}\cdot\ln\sqrt[3]{x}$
27	Расчет выражения: $u = \dfrac{\sin(a+b)x}{2\left(\sqrt{a}+x\right)} + \dfrac{\cos(a-b)y}{2(c-d)}$

28	Расчет выражения: $u = \frac{c}{a+b}\sin\left(\frac{a-b^2+c^3}{\sqrt{a-b}}\right)$
29	Расчет выражения: $u = \frac{\frac{1}{2ab}\ln\left(\sqrt{c^2-b^2}\cdot tgax+2\right)}{\sqrt{c^2-b^2}\cdot tgax-2}$
30	Расчет выражения: $u = \frac{\sqrt{y}}{2x^2}+\frac{1}{2a}\ln\frac{a+\sqrt{y}}{x}$
31	Расчет выражения: $u = \frac{\sin x}{\sqrt{3a}}+\frac{\cos y}{15}tg\frac{\ln a}{b}$
32	Расчет выражения: $u = \frac{\frac{\sin ax}{2a\cos^2 x}+\frac{1}{2a}\ln ax}{2}$
33	Расчет выражения: $u = \frac{\frac{10x-\frac{7}{12}y}{18z+\sqrt{x}}+\frac{1}{2b}\cos b}{5}$
34	Расчет выражения: $u = \frac{2mn}{m^3+n^3}+\frac{2m}{\sqrt{m-n}}-\sin\frac{1}{mn}$
35	Расчет выражения: $u = \frac{m+2}{ab}+\frac{\sqrt{2m}}{4}-\cos\frac{5}{am}$
36	Расчет выражения: $u = \frac{2x}{a^2}\sin ax-\left(\frac{x^2}{a}-\frac{2}{a^2}\right)\cdot\cos ax$
37	Расчет выражения: $u = \frac{\cos(x+y)a}{\sqrt{c}+\sqrt{3x}}+\frac{\sin(y-x)c}{xy}$
38	Расчет выражения: $u = \frac{1}{\sqrt{ab}}+tg\frac{c^2+2c-4a}{\sqrt{x}c}$
39	Расчет выражения: $u = \frac{\frac{2}{\sqrt{ay}}\ln\left(\sqrt{ax}+\sqrt{by}\right)}{a+y\cos b}$
40	Расчет выражения: $u = \frac{\frac{d-1}{\sqrt{a}}-\frac{d^2}{2c-a}\cdot\sqrt{\frac{b}{2c}}}{b^2-\sqrt{\frac{c}{d}\sin a}}$
41	Расчет выражения: $u = \frac{\frac{tgax}{2\cos^2 x}+\frac{1}{\sqrt{2a}}\ln\left(ax^{-2}\right)}{6\cos a+tg\frac{a}{x}}$
42	Расчет выражения: $u = \frac{a}{b}\ln\lvert c\rvert-\left(\frac{ax^2}{cy}-\frac{1}{\sqrt{x^3}}\right)\cos a$
43	Расчет выражения: $u = \frac{\frac{x}{2}+\frac{1}{2a}\ln(\sin ax+\cos ax)}{ctgax+\sqrt{a^2-x^a}}$

44	Расчет выражения: $u = \dfrac{axtg\dfrac{\sqrt{x\ -8}}{a^3+8\cos x}}{x+y\dfrac{\sin a - xa}{x^3-ay}}$
45	Расчет выражения: $u = \dfrac{a^3-x}{4a^2+x} - \dfrac{6}{a-\sqrt{3b}} + \ln\sqrt[5]{y}$
46	Расчет выражения: $u = \dfrac{2+tg^2 5x - \dfrac{\cos ay}{\sqrt{ax+10b^2}}}{\sqrt{\dfrac{x^2+\sin ay}{\cos ax - y^2}}}$
47	Расчет выражения: $u = \dfrac{\dfrac{\cos(a+b)x}{2(a+b)} + \dfrac{\cos(a-b)x}{2(a-b)}}{\sin(a+b)x - \sin(a-b)x}$
48	Расчет выражения: $u = \dfrac{ab}{\sqrt{x}} - \dfrac{c}{a=b+c-x}\ln\left(yx^2+\sqrt{c}\right)$
49	Расчет выражения: $u = \dfrac{\sqrt{4ax}}{b^2+a^2} + \dfrac{2a}{\sqrt{b-x}} - \ln\dfrac{1}{ax}$
50	Расчет выражения: $u = \dfrac{\dfrac{1}{\sqrt{b}}\ln\left(\sqrt{x}-\sqrt{b}\right)}{x+4b\cos(a-2b)}$

Таблица 6.2.

Индивидуальные варианты заданий для выполнения второй части работы

№ варианта	Задание
1	Нарушение ограничения диапазона допустимых значений для элементов массива
2	Треугольник с заданными сторонами не может существовать
3	Файл с таким именем уже существует в заданной директории
4	Размер введенного в поле текста больше нормы
5	Треугольник с заданными углами не может существовать
6	Отсутствует файл настроек приложения
7	Файл настроек имеет некорректный формат
8	Нарушение запрета на ввод цифр
9	Четырехугольник с заданными углами не может существовать
10	Нарушение запрета на ввод больших букв
11	Нарушение запрета на ввод пробелов
12	Точка не принадлежит прямой
13	Нарушение ограничения дискриминанта при вычислении корней квадратного уравнения
14	В введенном тексте отсутствуют буквы
15	Точка не принадлежит плоскости
16	Введено имя несуществующего файла

17	Нарушение запрета на ввод кириллицы
18	Доступ к приложению запрещен (пользователь ввел некорректные значения логин и пароль)
19	Прямая не принадлежит плоскости
20	Нарушение запрета на создание файла в определенной папке
21	Использование больше четырех знаков после запятой
22	Векторы не коллинеарные
23	Удаление несуществующего файла
24	Запись в несуществующий элемент массива
25	Четырёхугольник с заданными сторонами не может существовать
26	Размер введенного в поле текста меньше нормы
27	Введена буква, а не цифра
28	Запрет на использование согласных букв
29	Значение переменной вышло за пределы указанного диапазона
30	Использование положительных чисел
31	Поле текста пусто
32	Использование отрицательных чисел
33	Размер массива превышает максимальный заданный
34	Изменение элемента массива по несуществующему индексу
35	Первое слово в тексте не с большой буквы
36	Переполнение поля ввода (более 20 символов)
37	Размер введенного в поле текста меньше 8 символов
38	Запрет на использование гласных букв
39	Размер файла превышает заданный
40	Нарушение запрета на ввод букв
41	Не верно введен логин (пользователь с указанным логином не зарегистрирован)
52	Права доступа к приложению ограничены (пользователь ввел “гостевые” логин и пароль)
43	Невозможно зарегистрировать пользователя с таким именем
44	В введенном тексте отсутствуют цифры
45	В тексте отсутствует необходимое слово
46	Нарушение запрета на ввод латиницы
47	Больше одного пробела между словами
48	Нарушение запрета на ввод маленьких букв
49	Не верно введен пароль
50	Введённые строки не равны

Лабораторная работа №7
Работа с графикой в Windows приложениях

Цель работы: приобрести навыки по работе с графикой в Windows-приложениях.

7.1. Основные теоретические сведения

Для работы с графикой в языке C++ изначально не существовало специальных средств или конструкций языка, так как данный язык разрабатывался во времена операционных систем консольного типа, преимущественно MS-DOS. Однако по мере развития операционных систем графический режим стал в них поддерживаться на постоянной основе, а позже и вовсе стал неотъемлемой частью современных ОС.

Именно поэтому возможности языка программирования C++ по части работы с графикой видоизменялись вслед за развитием операционной системы. Эти возможности не входили в базовое ядро языка, а были доступны в виде дополнительных библиотек. Так для MS-DOS существовала единственная на тот момент библиотека graphics.h, которая позволяла выводить графические изображения при максимальном разрешении в 800x600 пикселей и глубиной в 16 цветов. Позже данная библиотека была модернизирована, что позволило использовать уже 256 цветовых оттенков. Основной сложностью использования подобных библиотек была необходимость переключения ОС в графический режим, так как на тот момент ОС все еще работали в текстовом режиме. И только с выходом первой версии Windows – Windows 3.11 – необходимость переключения отпала, так как сама ОС уже была графической.

Сегодня под каждую операционную систему существуют десятки графических библиотек, написанных для C++, которые содержат необходимые функции или классы для комфортной работы с графикой. Это позволяет как проектировать приложения с графическим интерфейсом на C++, так и рисовать графические примитивы. Так например, известная библиотека QT позволяет разрабатывать графические оконные приложения и содержит в своем составе все необходимые классы для вывода привычных элементов управления, таких как кнопки, флажки, поля ввода и т.п. Другие библиотеки не так сложны как QT и позволяют организовать процесс рисования: вывод линий, окружностей, надписей.

Одной из самых простых библиотек, предназначенных для вывода графики, является библиотека TX Library, которую свободно можно скачать по ссылке (http://storage.ded32.net.ru/Lib/TX/TXUpdate/Doc/HTML.ru). По утверждению автора библиотеки, она как раз создавалась для помощи студентам в освоении графики. При всей своей простоте эта библиотека имеет немало достоинств, среди которых присутствуют возможности по управлению не только графикой, но и звуком, сетью, мышью и т.п. С помощью этой библиотеки можно не только визуализировать, но и озвучить разрабатываемую программу, например, создать простое игровое приложение. Именно поэтому

автор настоящих методических указаний рекомендует использовать ее при выполнении данной лабораторной работы.

Для того, чтобы подключить библиотеку, ее необходимо установить в какую-нибудь папку (распространяется она в виде самораспаковывающегося архива), после чего необходимо единственный файл TXLib.h скопировать в рабочую папку своего проекта. После этого библиотеку можно удалить – файл TXLib.h не является заголовочным, как можно было бы подумать, а содержит весь необходимый функционал этой библиотеки. Далее следует подключить его стандартным образом к своему проекту – добавить в менеджере файлов и прописать конструкцию `#include "TXLib.h"` в своем проекте.

Рассмотрим более детально основные возможности библиотеки. Для начала рисования необходимо создать графическое окно. Для этого служит функция txCreateWindow. В качестве параметров в эту функцию следует передать размеры окна. После того, как окно успешно создано, можно вызывать другие функции для рисования. Далее рассмотрим их краткое назначение.

`txClear (HDC dc=txDC())` – стирает холст текущим цветом заполнения.

`txSetPixel (double x, double y, COLORREF color, HDC dc=txDC())` – рисует пиксель (точку на экране).

`txGetPixel (double x, double y, HDC dc=txDC())` – возвращает текущий цвет точки (пикселя) на экране.

`txLine (double x0, double y0, double x1, double y1, HDC dc=txDC())` – рисует линию.

`txRectangle (double x0, double y0, double x1, double y1, HDC dc=txDC())` – рисует прямоугольник.

`txPolygon (const POINT points[], int numPoints, HDC dc=txDC())` – рисует ломаную линию или многоугольник.

`txEllipse (double x0, double y0, double x1, double y1, HDC dc=txDC())` – рисует эллипс.

`txCircle (double x, double y, double r)` – рисует окружность или круг.

`txArc (double x0, double y0, double x1, double y1, double startAngle, double totalAngle, HDC dc=txDC())` – рисует дугу эллипса.

`txPie (double x0, double y0, double x1, double y1, double startAngle, double totalAngle, HDC dc=txDC())` – рисует сектор эллипса.

`txChord (double x0, double y0, double x1, double y1, double startAngle, double totalAngle, HDC dc=txDC())` – рисует хорду эллипса.

`txFloodFill (double x, double y, COLORREF color=TX_TRANSPARENT, DWORD mode=FLOODFILLSURFACE, HDC dc=txDC())` – заливает произвольный контур текущим цветом заполнения.

`txTextOut (double x, double y, const char text[], HDC dc=txDC())` – рисует текст.

`txSelectFont (const char name[], double sizeY, double sizeX=-1, int bold=FW_DONTCARE, bool italic=false, bool underline=false, bool strikeout=false, double angle=0, HDC dc=txDC())` – выбирает текущий шрифт, его размер и другие атрибуты.

`txSetTextAlign (unsigned align=TA_CENTER|TA_BASELINE, HDC dc=txDC())` – устанавливает текущее выравнивание текста (влево/вправо/по центру).

`RGB (int red, int green, int blue)` – создает (смешивает) цвет из трех базовых цветов (компонент).

`txSetColor (COLORREF color, double thickness=1, HDC dc=txDC())` – устанавливает текущий цвет и толщину линий, цвет текста.

`txGetColor (HDC dc=txDC())` – возвращает текущий цвет линий и текста.

`txSetFillColor (COLORREF color, HDC dc=txDC())` – устанавливает текущий цвет заполнения фигур.

`txGetFillColor (HDC dc=txDC())` – возвращает текущий цвет заполнения фигур.

`txExtractColor (COLORREF color, COLORREF component)` – извлекает цветовую компоненту (цветовой канал) из смешанного цвета.

`txRGB2HSL (COLORREF rgbColor)` – преобразует цвет из формата RGB в формат HSL.

`txHSL2RGB (COLORREF hslColor)` – преобразует цвет из формата HSL в формат RGB.

Представленные функции далеко не все, которые есть в данной библиотеке, но их уже достаточно чтобы нарисовать большинство фигур. Если этого покажется недостаточно, рекомендуем ознакомиться с полным описанием функций библиотеки на сайте автора по вышеприведенной ссылке или в конспекте лекций.

Другой популярной задачей является ввод каких либо параметров перед запуском или во время работы приложения при помощи диалогового окна. Данная библиотека умеет и это. Для создания диалогового окна необходимо унаследовать класс от класса txDialog, входящего в состав библиотеки. После чего потребуется создать карту компонентов (картой компонентов называется массив элементов, которые будут выводиться в диалоговом окне) и закрепить обработчики на нажатие соответствующих кнопок в создаваемом диалоге. Обработчиком считается специальный метод, которому передается управление тогда, когда пользователь выполняет действия с диалогом. Библиотека TX Library устроена так, что на любые действия пользователя вызывается один и тот же метод с именем dialogProc. Узнать тип возникшего события можно при помощи передаваемых в метод параметров. Более подробно данный механизм описан в документации к библиотеке и частично в комментариях к программному коду к примеру выполнения задания (см. раздел 7.3 далее).

Для реализации приведенных ниже заданий во многих вариантах потребуются также формулы поворота системы координат из области элементарной математики, так как потребуется вычислять новые координаты точек после поворота. Поэтому приведем их в данном разделе.

$$x = x'\cos a - y'\sin a$$

$$y = x'\sin a + y'\cos a$$

где x,y – координаты точки в новой системе координат, x',y' – координаты точки в старой системе координат, a – угол поворота.

7.2. Задание к лабораторной работе

Разработать оконное приложение для рисования фрактала – графического объекта, отдельные части которого в точности совпадают с самим собой.

В первую очередь, следует **установить и настроить** графическую библиотеку TX Library. Ее можно взять у преподавателя или скачать самостоятельно (http://storage.ded32.net.ru/Lib/TX/TXUpdate/Doc/HTML.ru/). На момент издания настоящих методических указаний (01.02.2022) ссылка для скачивания актуальна. *Допускается использование и других графических библиотек для языка C++ аналогичного назначения*.

Примечание: перед выполнением основного задания можно оценить работу графической библиотеки, запустив примеры кода, которые есть в разделе документации по выше приведенной ссылке.

Далее следует проанализировать структуру фрактала и выделить общий элемент рисунка, после чего написать функцию, которая рисует этот элемент. Функция должна принимать параметры – это могут быть, например, размеры элемента или координаты начальной точки для рисования, или угол поворота элемента и т.п. Далее созданную функцию вызывают рекурсивно, причем **обязательно необходимо предусмотреть условие выхода**. В противном случае приложение зависнет.

Рисование фрактала обычно выполняется по уровням: сначала рисуется объект верхнего уровня, затем к нему дорисовываются объекты последующих уровней, имеющие, как правило, общие точки с объектом верхнего уровня, но меньшие размеры. Однако, в некоторых вариантах заданий (см. например, вариант 13) рисование должно начинаться «с конца», т.е. самого нижнего уровня объекта.

Рисование фрактала должно прекращаться если:

– объект текущего уровня становится соизмеримым с размерами одного пикселя окна;

– объект текущего уровня становится соизмеримым с заданными пользователем размерами (определяется параметрами);

– достигнуто необходимое количество уровней (определяется параметрами).

Помимо фрактала необходимо **вывести подпись – ФИО студента и группу** в произвольном месте изображения.

Индивидуальные варианты заданий приведены в разделе 7.6. Перед рисованием фрактала **пользователь должен указать начальные параметры**, приведенные в каждом варианте. Ввод исходных данных лучше всего предусмотреть прямо в созданном окне (библиотека TX Library поддерживает диалоговые окна, при помощи которых можно организовать ввод данных). Если это по какой либо причине невозможно, можно организовать ввод данных из консоли (перед открытием основного окна приложения).

7.3. Пример выполнения лабораторной работы

Реализовать приложение для рисования фрактала «Парус». В программе должен быть предусмотрен ввод следующих параметров: X,Y – координаты начальной точки, Width – ширина квадрата, Coef – коэффициент уменьшения ширины квадрата на последующем уровне, Level – максимальное число уровней.

Файл inputDialog.h

```
#include "TXLib.h"
#define IDX 200 /* Id полей ввода */
#define IDY 201
#define IDW 202
#define IDL 203
#define IDC 204
class inputDialog : public txDialog
{
    private:
        // Параметры
        int x, y, width, level;
        double coef;
        // Результат работы диалога
        int result; // 0 - диалог открыт, пользователь его еще не закрыл
                    // IDOK - нажата кнопка OK
                    // IDCANCEL - нажат крестик
    public:
        // Конструктор, задает значения по умолчанию
        inputDialog();
        // Геттеры, получают значения параметров
        int getX();
        int getY();
        int getWidth();
        int getLevel();
        double getCoef();
        int getResult();
        // Обработчик события нажатия на кнопку
        int dialogProc(HWND wnd, UINT msg, WPARAM wParam, LPARAM lParam);
        // Показывает диалог
        void dialogBox();
};
```

Файл inputDialog.cpp

```
#include "inputDialog.h"
// Конструктор, задает значения по умолчанию
inputDialog::inputDialog()
{
    this -> x = 400;
    this -> y = 550;
    this -> width = 90;
    this -> level = 6;
    this -> coef = 0.7;
    this -> result = 0;
}
// Геттеры, получают значения параметров
int inputDialog::getX()
{
```

```
    return this -> x;
}
int inputDialog::getY()
{
    return this -> y;
}
int inputDialog::getWidth()
{
    return this -> width;
}
int inputDialog::getLevel()
{
    return this -> level;
}
double inputDialog::getCoef()
{
    return this -> coef;
}
int inputDialog::getResult()
{
    return this -> result;
}
// Обработчик события нажатия на кнопку
int inputDialog::dialogProc(HWND wnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    // Зафиксировано нажатие кнопки
    if (lParam && msg == WM_COMMAND && HIWORD(wParam) == BN_CLICKED)
    {
        // Получить id нажатой кнопки
        int id = LOWORD(wParam);
        // Если нажата кнопка OK - переписать параметры
        // из текстовых полей в свойства класса
        if (id == IDOK)
        {
            char strX[32], strY[32], strW[32], strL[32], strC[32];
            GetDlgItemText(wnd, IDX, strX, sizeof(strX) - 1);
            GetDlgItemText(wnd, IDY, strY, sizeof(strY) - 1);
            GetDlgItemText(wnd, IDW, strW, sizeof(strW) - 1);
            GetDlgItemText(wnd, IDL, strL, sizeof(strL) - 1);
            GetDlgItemText(wnd, IDC, strC, sizeof(strC) - 1);
            this -> x = atoi(strX);
            this -> y = atoi(strY);
            this -> width = atoi(strW);
            this -> level = atoi(strL);
            this -> coef = atof(strC);
        }
        // Если нажата кнопка OK или крестик
        if (id == IDOK || id == IDCANCEL)
        {
            // Сохранить результат работы диалога
            this -> result = id;
            // Закрыть окно
            DestroyWindow(wnd);
        }
    }
    return 0;
}
// Показывает диалог
void inputDialog::dialogBox()
{
    // Значения параметров
    string strX = std::to_string(this -> x);
    string strY = std::to_string(this -> y);
    string strW = std::to_string(this -> width);
```

```
    string strL = std::to_string(this -> level);
    string strC = std::to_string(this -> coef);

    // Стили для элементов
    DWORD EditStyles = ES_LEFT | WS_BORDER | ES_AUTOHSCROLL | WS_TABSTOP;
    DWORD StaticStyles = SS_LEFT;
    DWORD ButtonOkStyles = BS_DEFPUSHBUTTON | WS_TABSTOP;
    // Карта элементов диалога
    txDialog::Layout layout[] = {
        { txDialog::DIALOG, "Ввод параметров",
          5, 0, 0, 200, 210 },
        { txDialog::STATIC, "Координата X",
          101, 10, 10, 180, 10, StaticStyles },
        { txDialog::EDIT, strX.c_str(),
          IDX, 10, 20, 180, 15, EditStyles },
        { txDialog::STATIC, "Координата Y",
          102, 10, 45, 180, 10, StaticStyles },
        { txDialog::EDIT, strY.c_str(),
          IDY, 10, 55, 180, 15, EditStyles },
        { txDialog::STATIC, "Ширина квадрата",
          103, 10, 80, 180, 10, StaticStyles },
        { txDialog::EDIT, strW.c_str(),
          IDW, 10, 90, 180, 15, EditStyles },
        { txDialog::STATIC, "Уровень",
          104, 10, 115, 180, 10, StaticStyles },
        { txDialog::EDIT, strL.c_str(),
          IDL, 10, 125, 180, 15, EditStyles },
        { txDialog::STATIC, "Коэффициент сжатия",
          105, 10, 150, 180, 10, StaticStyles },
        { txDialog::EDIT, strC.c_str(),
          IDC, 10, 160, 180, 15, EditStyles },
        { txDialog::BUTTON, "&OK",
          IDOK, 75, 185, 50, 15, ButtonOkStyles },
        { txDialog::END }
    };
    // Установить карту элементов
    this -> setLayout(layout);
    // Показать диалоговое окно
    txDialog::dialogBox();
}
```

Файл lab7.cpp

```
/* Фрактал Парус */
#include "TXLib.h"
#include "inputDialog.h"
// Глобальная переменная, определяющая уровень построения
int LEVEL = 1;
// Функция, которая рисует текущий уровень в виде
//   ________
//  |        |
//  |________|
//       |
//       |
// x,y - координаты начальной точки
// dir - направление поворота (0-3)
// width - ширина квадрата (и высота линии)
// coef - коэффициет сжатия
// level - уровень прорисовки
void Sail(int x, int y, int dir, int width, double coef, int level)
{
    // Координаты конечной точки
    int x1 = 0, y1 = 0;
```

```
// Координаты двух вершин квадрата
int qx1 = 0, qy1 = 0, qx2 = 0, qy2 = 0;
// Координаты начальных точек для оставшихся трех сторон квадрата
int nx0 = 0, ny0 = 0, nx1 = 0, ny1 = 0, nx2 = 0, ny2 = 0;
// Направления поворота элементов следующего уровня
int dir0 = 0, dir1 = 0, dir2 = 0;
// Условия выхода
// Если размер элемента соизмерим с размером одного пикселя - выход
if (width <= 1)
    return;
// Если уровень соответствует указанному пользователем - выход
if (level == LEVEL + 1)
    return;
// Вычислить координаты всех необходимых точек
// в зависимости от направления поворота
switch (dir)
{
case 0: // вверх
    x1 = x;
    y1 = y - width;
    qx1 = x - width / 2;
    qy1 = y - 2 * width;
    qx2 = x + width / 2;
    qy2 = y - width;
    break;
case 1: // вправо
    x1 = x + width;
    y1 = y;
    qx1 = x + width;
    qy1 = y - width / 2;
    qx2 = x + 2 * width;
    qy2 = y + width / 2;
    break;
case 2: // вниз
    x1 = x;
    y1 = y + width;
    qx1 = x - width / 2;
    qy1 = y + 2 * width;
    qx2 = x + width / 2;
    qy2 = y + width;
    break;
case 3: // влево
    x1 = x - width;
    y1 = y;
    qx1 = x - width;
    qy1 = y - width / 2;
    qx2 = x - 2 * width;
    qy2 = y + width / 2;
    break;
}
// Убрать цвет заливки, который установлен по умолчанию
txSetFillColor(TX_NULL);
// Нарисовать линию
txLine(x, y, x1, y1);
// Нарисовать квадрат
txRectangle(qx1, qy1, qx2, qy2);
// Вычислить координаты начальных точек для
// рисования следующего уровня и направление
// поворота элемента следующего уровня
switch (dir)
{
case 0: // вверх
    nx0 = x - width / 2;
    ny0 = y - width - width / 2;
```

```
            nx1 = x;
            ny1 = y - 2 * width;
            nx2 = x + width / 2;
            ny2 = y - width - width / 2;
            dir0 = 3;
            dir1 = 0;
            dir2 = 1;
            break;
        case 1: // вправо
            nx0 = x + width + width / 2;
            ny0 = y - width / 2;
            nx1 = x + 2 * width;
            ny1 = y;
            nx2 = x + width + width / 2;
            ny2 = y + width / 2;
            dir0 = 0;
            dir1 = 1;
            dir2 = 2;
            break;
        case 2: // вниз
            nx0 = x + width / 2;
            ny0 = y + width + width / 2;
            nx1 = x;
            ny1 = y + 2 * width;
            nx2 = x - width / 2;
            ny2 = y + width + width / 2;
            dir0 = 1;
            dir1 = 2;
            dir2 = 3;
            break;
        case 3: // влево
            nx0 = x - width - width / 2;
            ny0 = y + width / 2;
            nx1 = x - 2 * width;
            ny1 = y;
            nx2 = x - width - width / 2;
            ny2 = y - width / 2;
            dir0 = 2;
            dir1 = 3;
            dir2 = 0;
            break;
        }
        // Вызвать данную функцию рекурсивно 3 раза
        // для каждой стороны квадрата
        Sail(nx0, ny0, dir0, (int)(width*coef), coef, level + 1);
        Sail(nx1, ny1, dir1, (int)(width*coef), coef, level + 1);
        Sail(nx2, ny2, dir2, (int)(width*coef), coef, level + 1);
    }
    // Главная функция
    void main()
    {
        // Ввод параметров
        inputDialog dlg;
        dlg.dialogBox();
        if (dlg.getResult() == IDOK)
        {
            // Создать окно
            txCreateWindow(800, 700);
            LEVEL = dlg.getLevel();
            Sail(dlg.getX(), dlg.getY(), 0, dlg.getWidth(), dlg.getCoef(), 1);
        }
    }
```

Результат работы программы при параметрах: X=400, Y=550, Width=70, Coef=0.9, Level=3 показан на рис. 7.3.1.

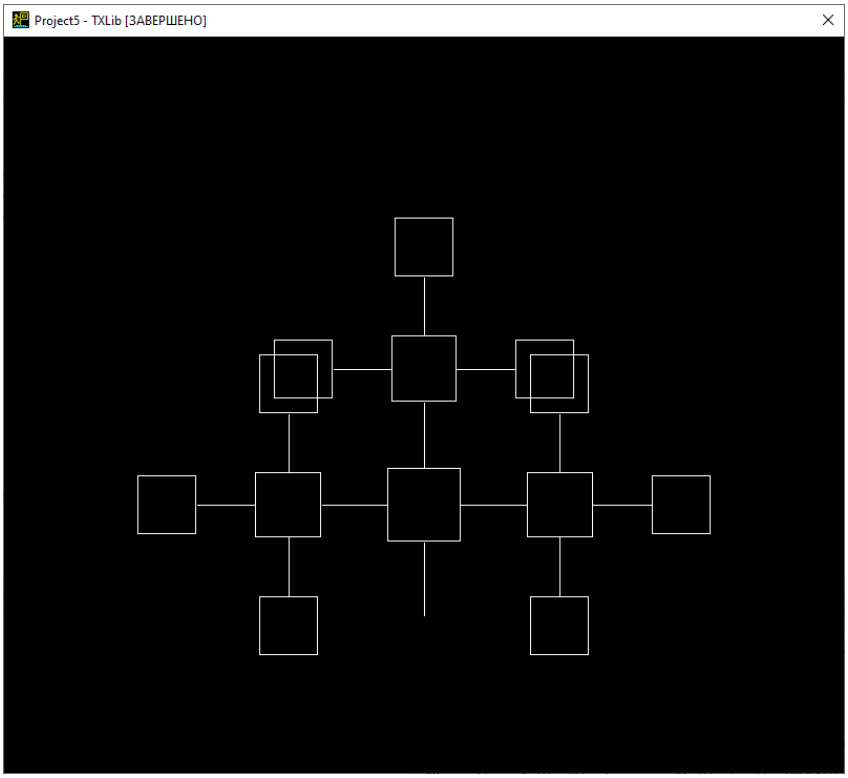


Рис. 7.3.1. – Результат работы программы (тест 1)

Результат работы программы при параметрах: X=400, Y=550, Width=90, Coef=0.7, Level=6 показан на рис. 7.3.2.

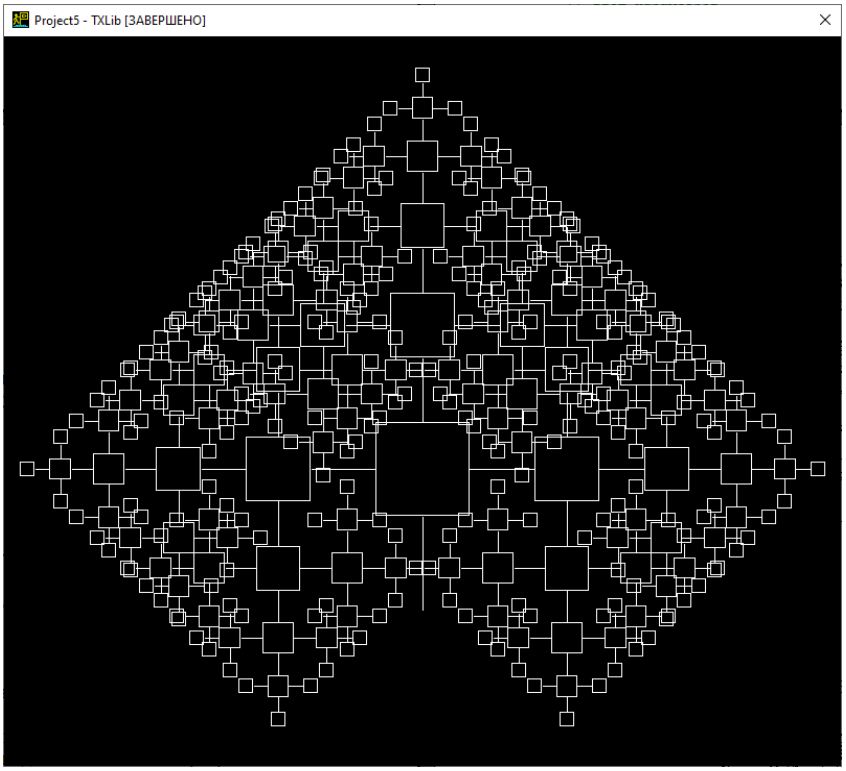


Рис. 7.3.2. – Результат работы программы (тест 2)

7.4. Содержание отчета

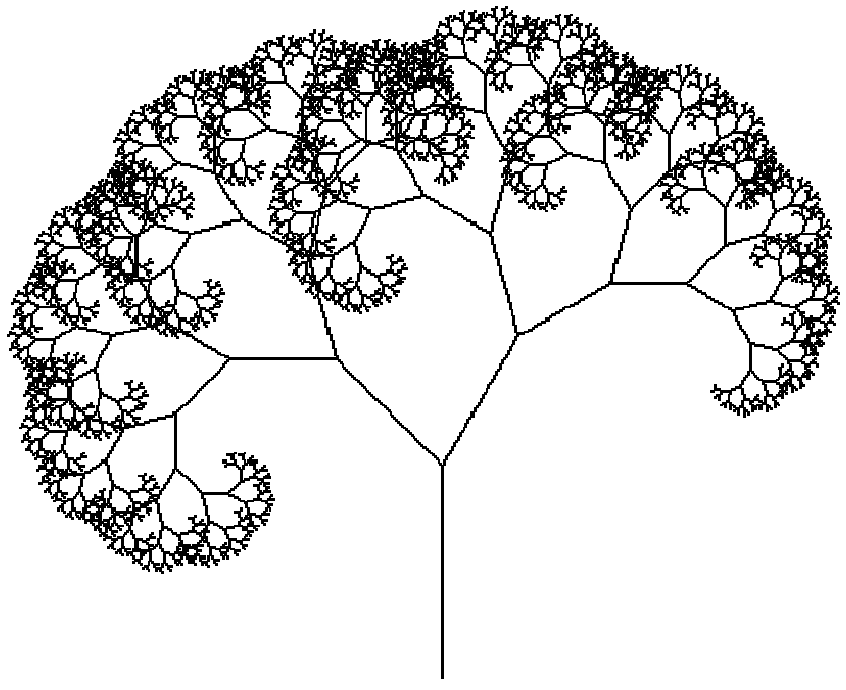
1. Титульный лист.
2. Условие лабораторной работы.
3. Текст программы.
4. Экранные формы с примерами работы программы.

7.5. Контрольные вопросы

1. Как установить параметры выводимой линии (цвет, толщину, стиль)?
2. Какая функция используется для заливки замкнутой области?
3. Какая функция позволяет вывести текст?
4. Каким образом можно создать диалоговое окно в программе?
5. Каким образом избежать мигания изображения при перерисовке?

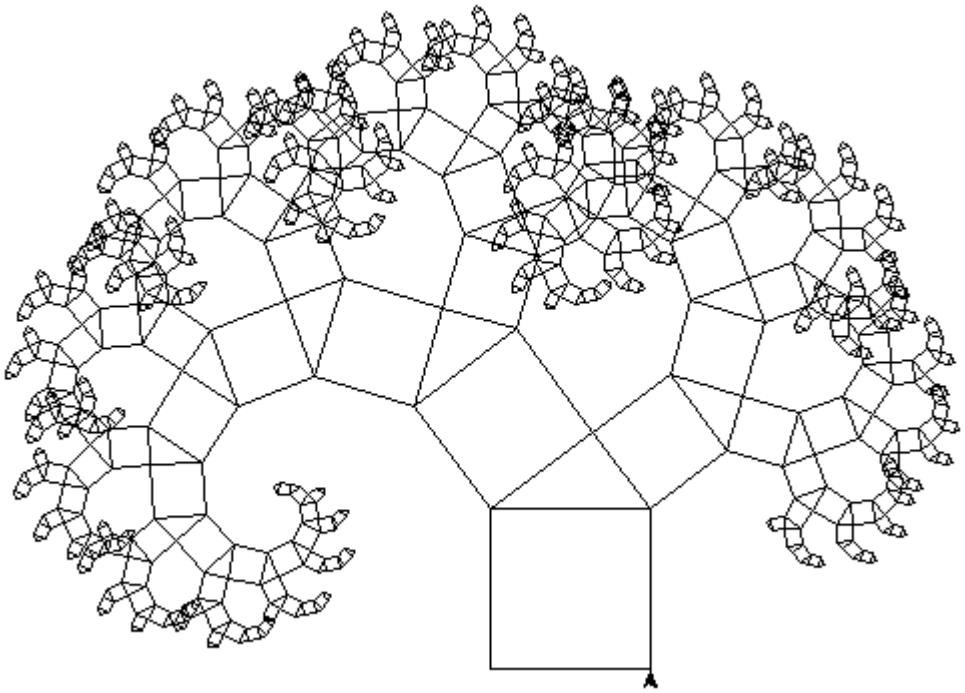
7.6. Варианты заданий

Индивидуальные варианты заданий изображены на рис. 7.6.1 – 7.6.50 (номер рисунка соответствует номеру варианта). В некоторых вариантах представлен вид фрактала на нескольких уровнях. Все углы в заданиях задаются в градусах, размеры – в пикселях.



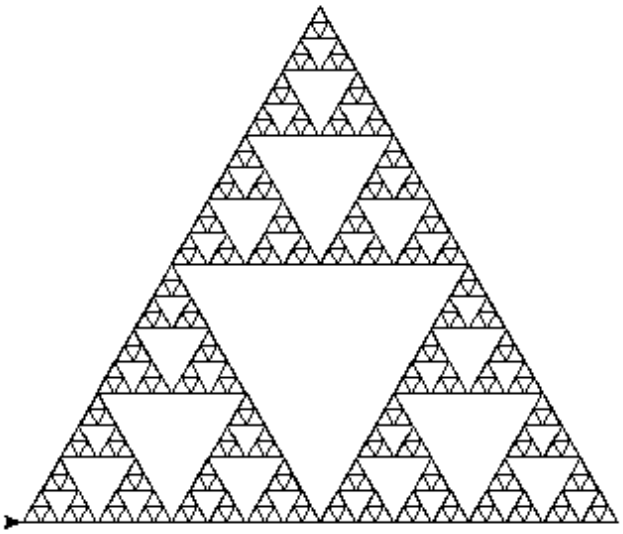
Параметры: X,Y – координаты начальной точки (внизу в центре), L – длина образующего отрезка (внизу в центре), A – угол между отрезками, K – коэффициент уменьшения длины отрезка на последующем уровне, Level – максимальное число уровней.

Рис. 7.6.1. – Фрактал «Восхождение» (вариант 1).



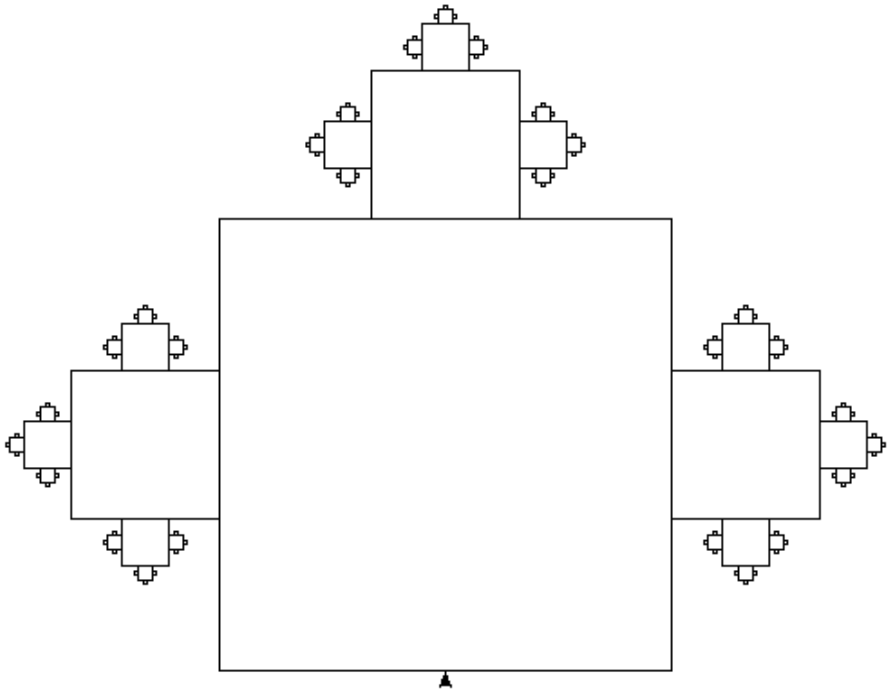
Параметры: X,Y – координаты правого нижнего угла квадрата (отмечено стрелкой), L – длина стороны квадрата (отмечено стрелкой), Level – максимальное число уровней.

Рис. 7.6.2. – Фрактал «Дерево Пифагора 1» (вариант 2).



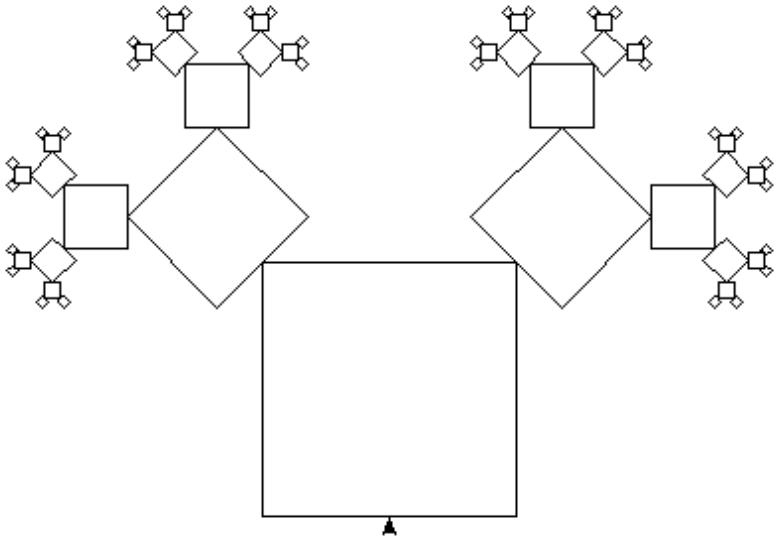
Параметры: X,Y – координаты вершины треугольника (отмечено стрелкой), L – длина стороны треугольника, Level – максимальное число уровней.

Рис. 7.6.3. – Фрактал «Треугольник Серпинского 1» (вариант 3).



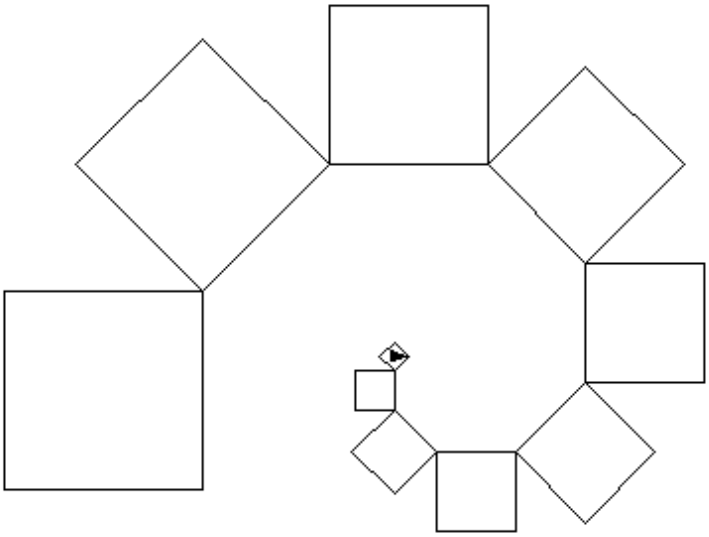
Параметры: X,Y – координаты середины нижней стороны квадрата (отмечено стрелкой), L – длина стороны квадрата, K – коэффициент уменьшения длины стороны квадрата на последующем уровне, Level – максимальное число уровней.

Рис. 7.6.4. – Фрактал «Квадратное растение» (вариант 4).



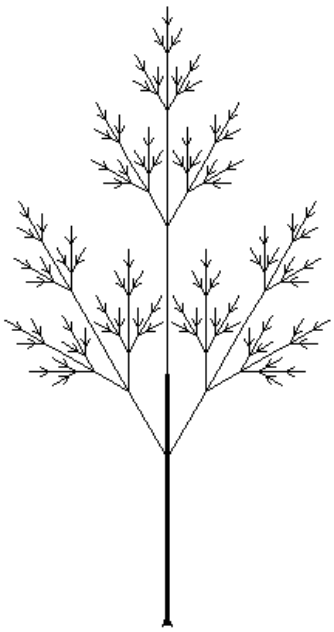
Параметры: X,Y – координаты середины нижней стороны квадрата (отмечено стрелкой), L – длина стороны квадрата, K – коэффициент уменьшения длины стороны квадрата на последующем уровне, Level – максимальное число уровней. В данном варианте все углы равны 45°.

Рис. 7.6.5. – Фрактал «Квадратное дерево» (вариант 5).



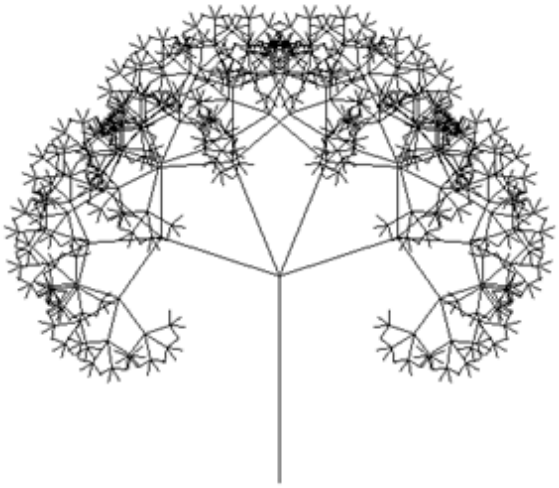
Параметры: X,Y – координаты правой вершины квадрата (отмечено стрелкой), L – длина стороны квадрата (отмечено стрелкой), A – угол поворота квадратов, K – коэффициент увеличения (>1) стороны квадрата на последующем уровне, Level – максимальное число уровней.

Рис. 7.6.6. – Фрактал «Квадратная спираль» (вариант 6).



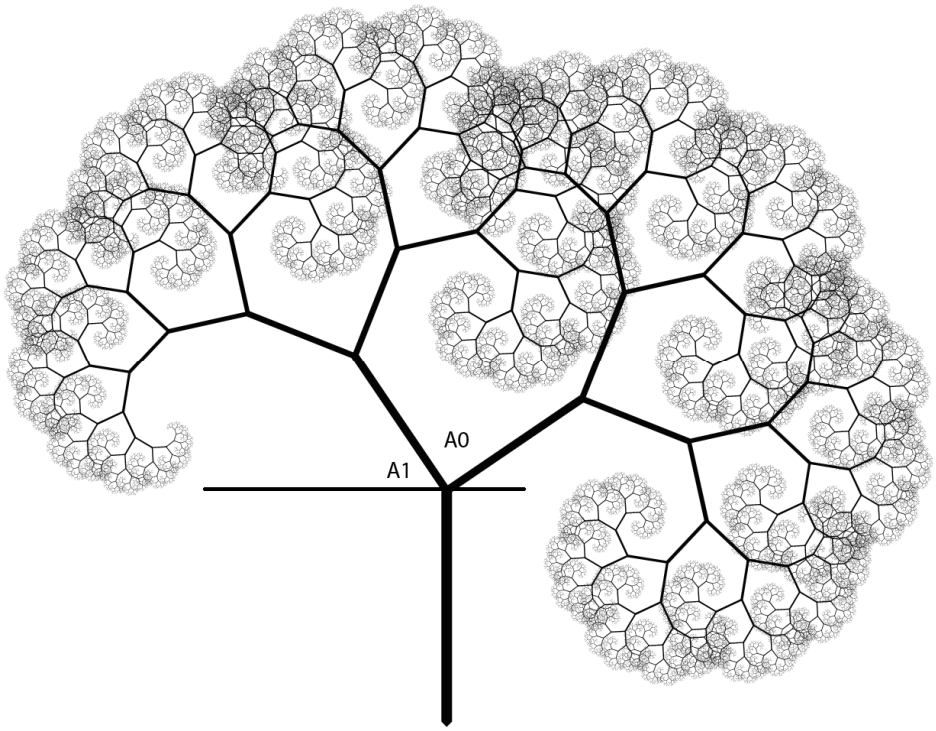
Параметры: X,Y – координаты начальной точки (отмечено стрелкой), L – длина образующего отрезка (показано жирным), A – угол между ветвями, K – коэффициент уменьшения длины отрезка на последующем уровне, C – коэффициент, определяющий отношение частей образующего отрезка, Level – максимальное число уровней.

Рис. 7.6.7. – Фрактал «Хвощ полевой» (вариант 7).



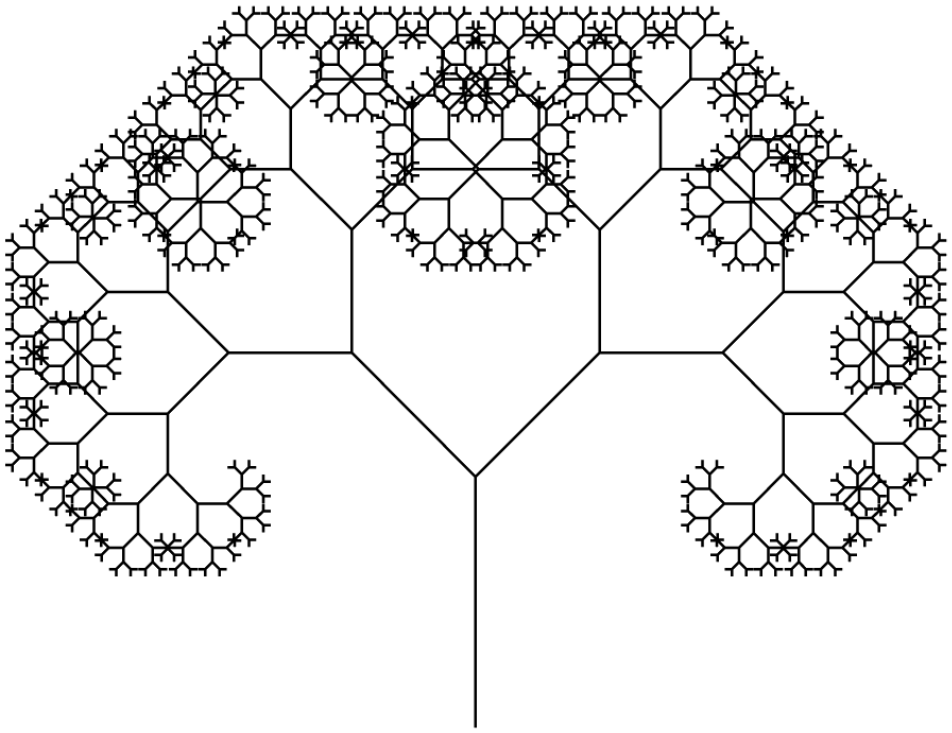
Параметры: X,Y – координаты начальной точки, L – длина образующего отрезка, A – угол между ветвями, K – коэффициент уменьшения длины отрезка на последующем уровне, Level – максимальное число уровней.

Рис. 7.6.8. – Фрактал «Равностороннее дерево» (вариант 8).



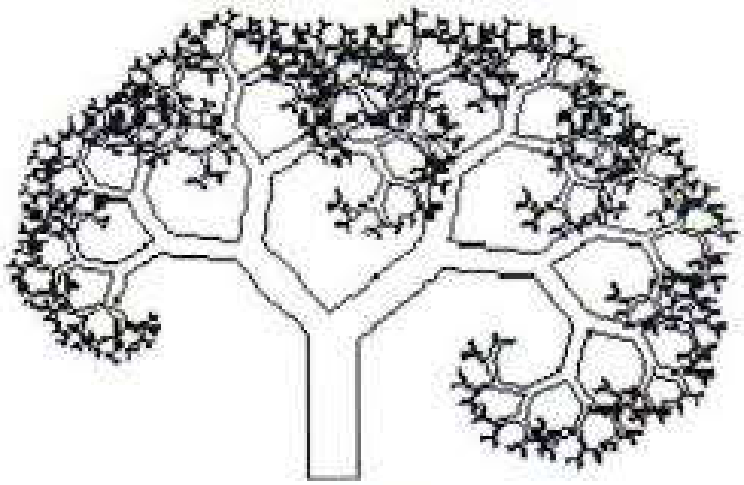
Параметры: X,Y – координаты начальной точки, L – длина образующего отрезка, W – толщина образующего отрезка, A0 – угол между ветвями, A1 – угол между первой ветвью и перпендикулярной образующему отрезку прямой, L0 – коэффициент уменьшения длины отрезка на последующем уровне, W0 – коэффициент уменьшения толщины отрезка на последующем уровне, Level – максимальное число уровней.

Рис. 7.6.9. – Фрактал «Клен 1» (вариант 9).



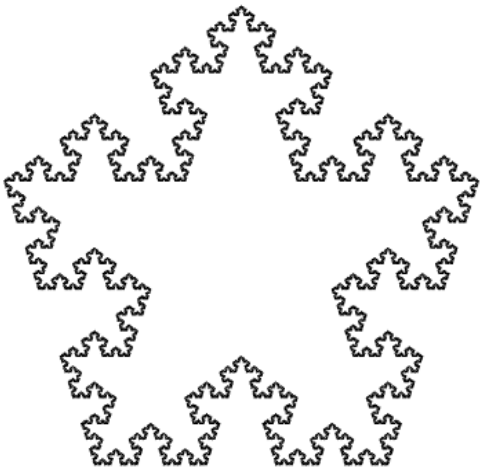
Параметры: X,Y – координаты начальной точки, L – длина образующего отрезка, K – коэффициент уменьшения длины отрезка на последующем уровне, Level – максимальное число уровней. В данном варианте все углы между ветвями равны 90°.

Рис. 7.6.10. – Фрактал «Дерево Пифагора 2» (вариант 10).



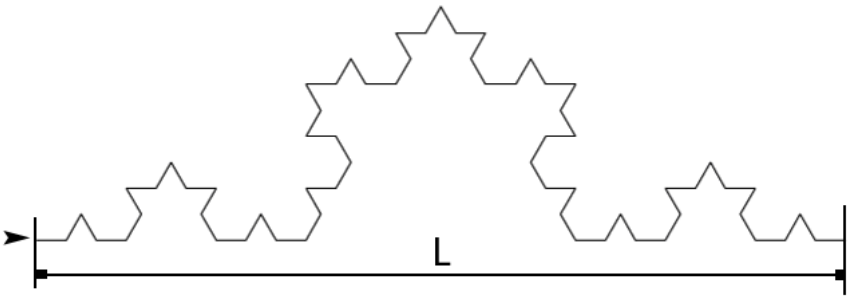
Параметры: X,Y – координаты начальной точки (середина ствола дуба), L – длина ствола, A0 – угол между ветвями, A1 – угол между первой ветвью и перпендикулярной образующему отрезку прямой (см. пояснения к рис. 7.6.9), L0 – коэффициент уменьшения длины ветви на последующем уровне, W0 – коэффициент уменьшения толщины ветви на последующем уровне, Level – максимальное число уровней.

Рис. 7.6.11. – Фрактал «Дуб» (вариант 11).



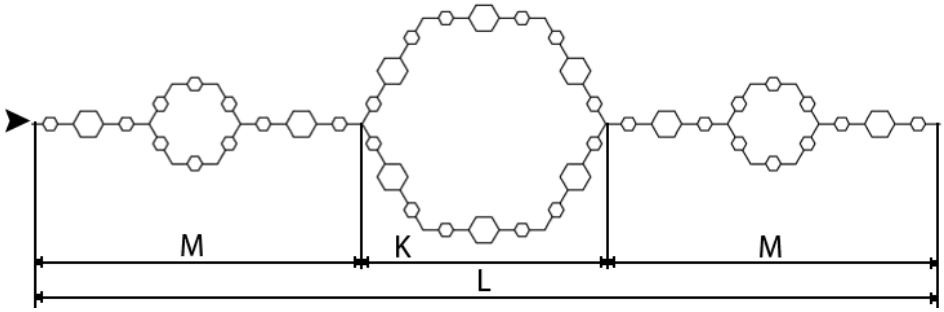
Параметры: X,Y – координаты центра пентаграммы, R – радиус пентаграммы, Level – максимальное число уровней.

Рис. 7.6.12. – Фрактал «Снежинка Коха» (вариант 12).



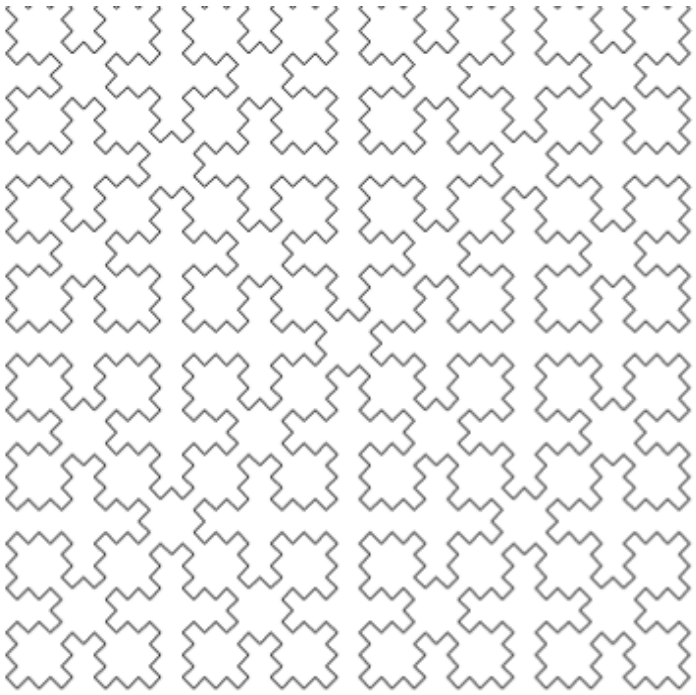
Параметры: X,Y – координаты начальной точки (левая сторона кривой, отмечена стрелкой), L – длина образующего отрезка, Level – максимальное число уровней.

Рис. 7.6.13. – Фрактал «Триадная кривая Коха» (вариант 13).



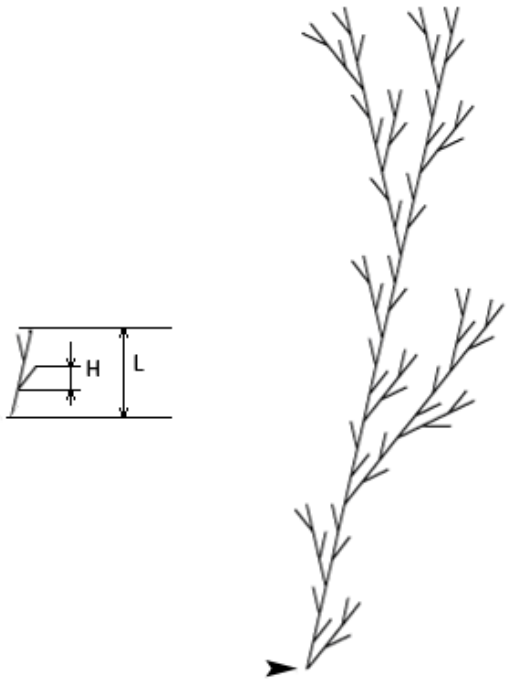
Параметры: X,Y – координаты начальной точки (левая сторона цепочки, отмечена стрелкой), L – длина цепочки, Coef – коэффициент, показывающий отношение размеров центрального звена цепочки к ее краям, т.е. Coef = K/M, L=2*M+K. Параметры M и K на чертеже показаны для пояснения, вводить в программе их не нужно. Level – максимальное число уровней.

Рис. 7.6.14. – Фрактал «Цепочка» (вариант 14).



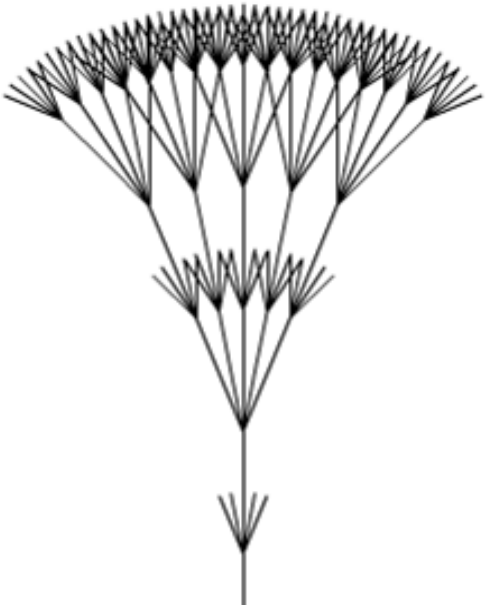
Параметры: W – размер квадрата, Level – максимальное число уровней.

Рис. 7.6.15. – Фрактал «Квадратная кривая Серпинского» (вариант 15).



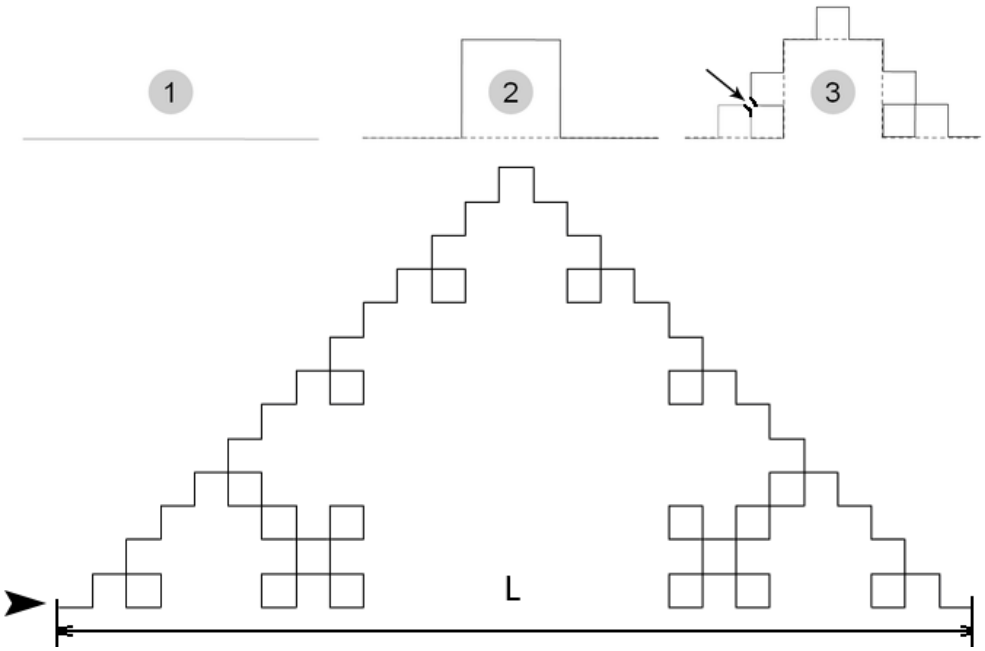
Параметры: X,Y – координаты начальной точки (внизу рисунка, отмечено стрелкой), L – длина стебля на текущем уровне, H – длина ветви на текущем уровне, A – угол наклона стебля относительно горизонта, B – угол наклона ветви относительно стебля, Level – максимальное число уровней. Слева показан образующий элемент фрактала (первый уровень).

Рис. 7.6.16. – Фрактал «Трава» (вариант 16).



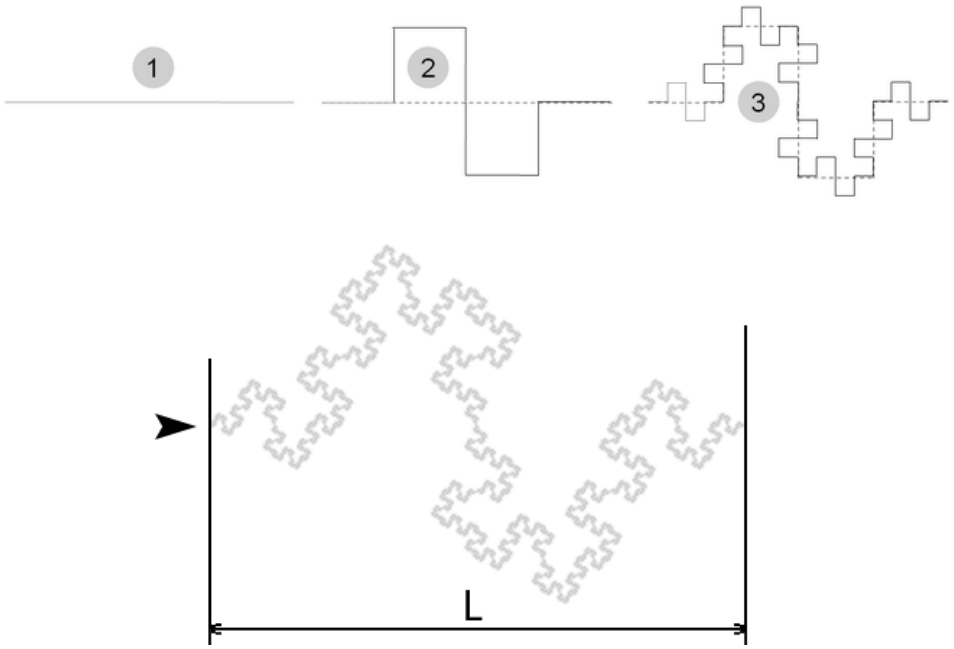
Параметры: X,Y – координаты начальной точки (внизу рисунка), L – длина стебля на текущем уровне, H – длина ветви на текущем уровне, A – угол между ветвями, Level – максимальное число уровней.

Рис. 7.6.17. – Фрактал «Цветок» (вариант 17).



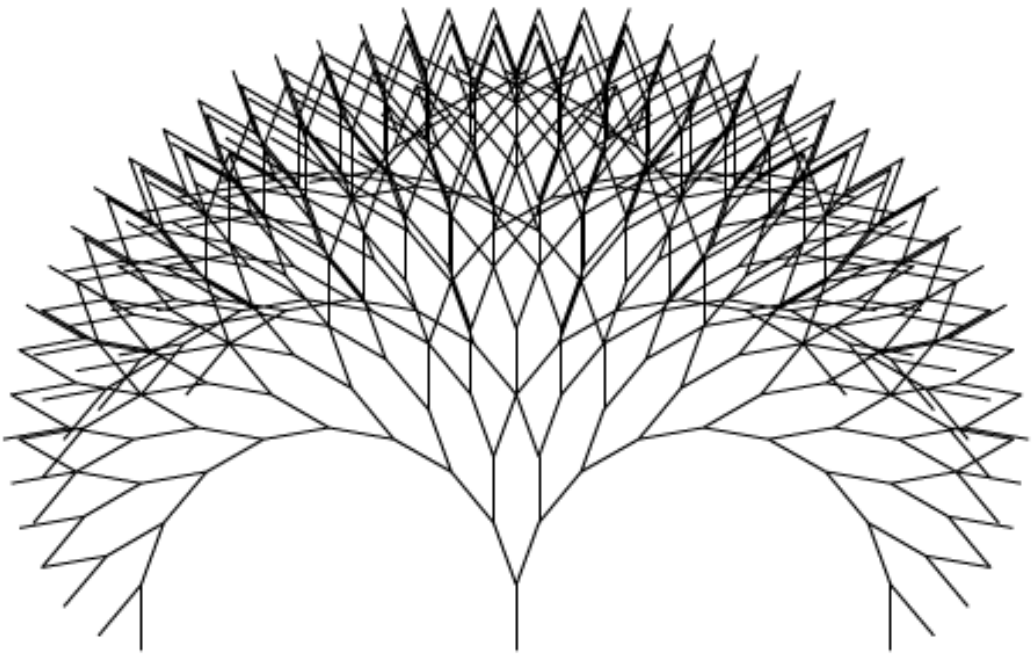
Параметры: X,Y – координаты начальной точки (внизу рисунка, отмечено стрелкой), L – длина образующего отрезка, Level – максимальное число уровней. Вверху рисунка показаны этапы построения. Обратите внимание на место, помеченное длинной стрелкой на уровне 3: здесь нет петли, как можно подумать, глядя на обобщенный рисунок, а просто смыкаются два П–образных элемента. Это ключевой факт, осознав который можно сильно упростить построение фрактала.

Рис. 7.6.18. – Фрактал «Квадратичная кривая Коха первого типа» (вариант 18).



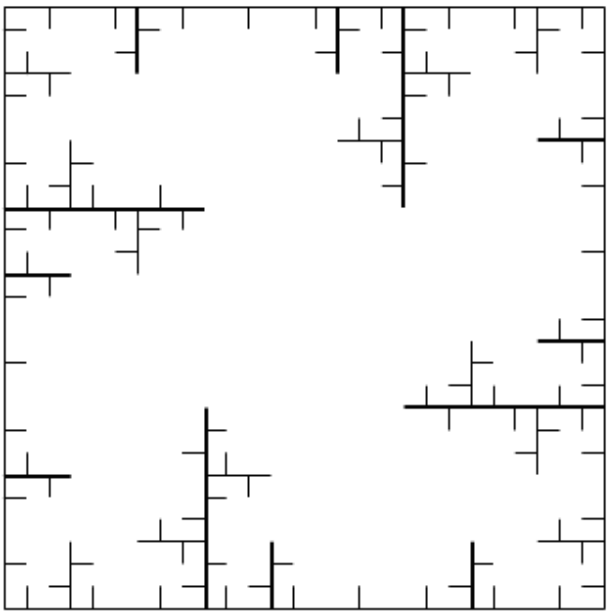
Параметры: X,Y – координаты начальной точки (внизу рисунка, отмечено стрелкой), L – длина образующего отрезка, Level – максимальное число уровней. Вверху рисунка показаны этапы построения.

Рис. 7.6.19. – Фрактал «Квадратичная кривая Коха второго типа (кривая Минковского)» (вариант 19).



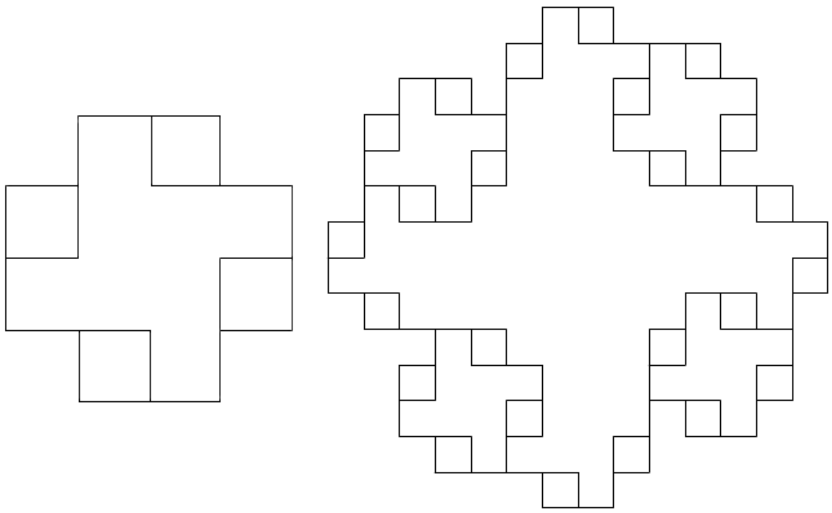
Параметры: X,Y – координаты начальной точки (внизу рисунка, в центре), L – длина стебля и ветви, A – угол между ветвями, Level – максимальное число уровней.

Рис. 7.6.20. – Фрактал «Шаровидный куст» (вариант 20).



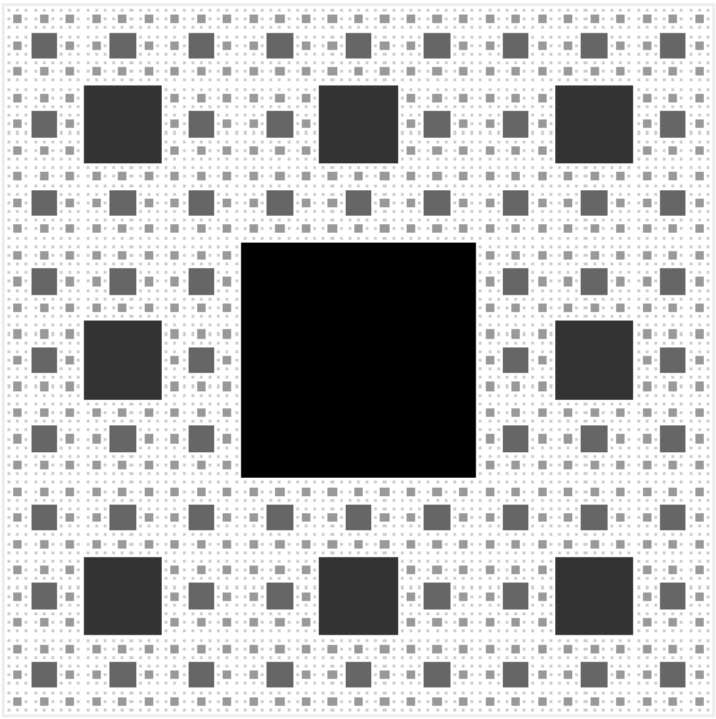
Параметры: X,Y – координаты левого нижнего угла квадрата, L – длина стороны квадрата, Level – максимальное число уровней. В данном варианте отношение образующих отрезков, имеющих общие точки со стороной квадрата, равно 9:27=1:3, 12:27 и 21:27 соответственно. Рисунок симметрично повторяется на всех четырех сторонах квадрата и на образующих отрезках предыдущего уровня. Для понимания образующие отрезки на первом уровне выделены жирным линией, на деле же все линии должны иметь одинаковую толщину в 1 пиксель.

Рис. 7.6.21. – Фрактал «Кристалл» (вариант 21).



Параметры: X,Y – координаты центра фигуры, L – ширина и высота фигуры, Level – максимальное число уровней. В данном варианте на первом уровне строится фигура, изображенная слева, на втором уровне к ее вершинам и квадратным элементам между вершинами достраиваются подобные фигуры, размер которых в 2 раза меньше и т.д. В конце необходимо удалить лишние линии.

Рис. 7.6.22. – Фрактал «Кольца» (вариант 22).

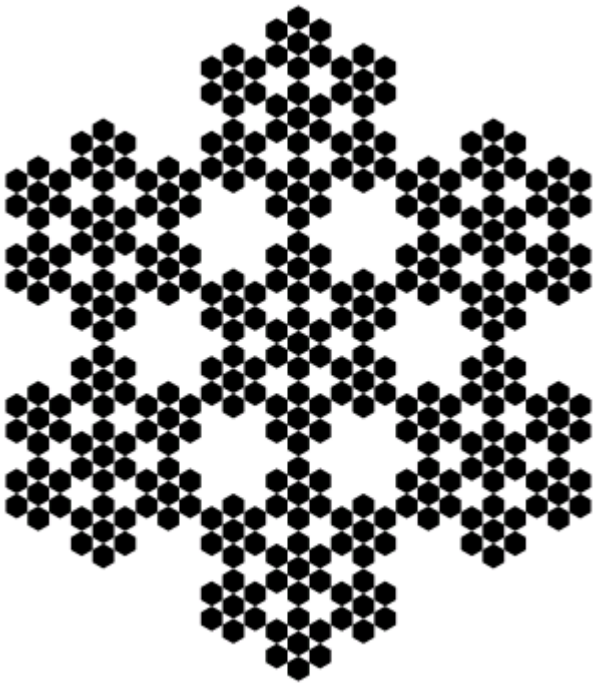


Параметры: X,Y – координаты центра квадрата, L – ширина и высота квадрата, K – коэффициент осветления квадратов последующих уровней, Level – максимальное число уровней. В данном варианте на первом уровне строится квадрат, размеры которого составляют L/3. После этого к каждой стороне и вершинам достраиваются квадраты, размер которых составляет 1/3 от первоначального квадрата, а значения RGB увеличиваются на K.

Рис. 7.6.23. – Фрактал «Квадрат Серпинского» (вариант 23).

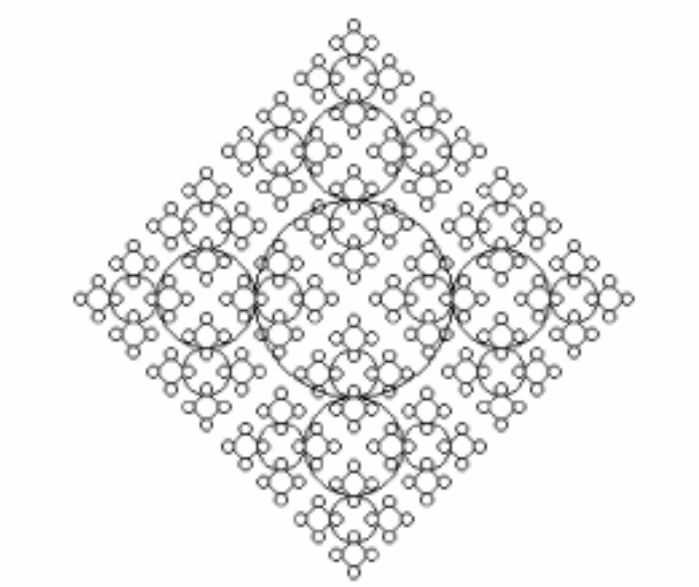
Параметры: X,Y – координаты центра пентаграммы первого уровня, R – радиус описанной окружности вокруг пентаграммы первого уровня, Level – максимальное число уровней. В данном варианте на первом уровне строится пентаграмма (правильный пятиугольник), после чего ко всем пяти ее сторонам достраиваются аналогичные пентаграммы. При построении следует учитывать, что закрашивание пентаграмм цветом следует производить только на последнем уровне.

Рис. 7.6.24. – Фрактал «Пятиугольная снежинка» (вариант 24).



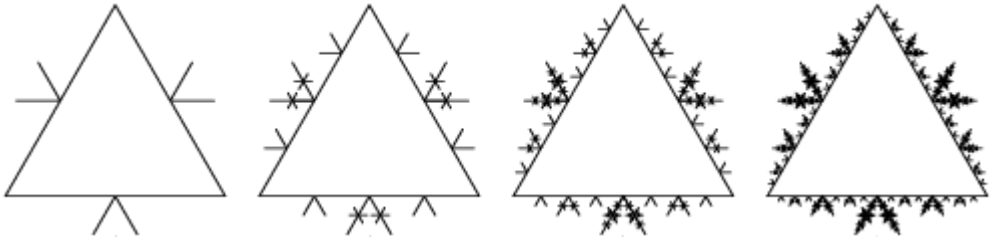
Параметры: X,Y – координаты центра шестиугольника первого уровня, R – радиус описанной окружности вокруг шестиугольника первого уровня, Level – максимальное число уровней. В данном варианте на первом уровне строится правильный шестиугольник, после чего ко всем пяти его вершинам достраиваются аналогичные шестиугольники. При построении следует учитывать, что закрашивание фрактала цветом следует производить только на последнем уровне.

Рис. 7.6.25. – Фрактал «Шестиугольная снежинка» (вариант 25).



Параметры: X,Y – координаты центра фигуры, R – радиус окружности первого уровня, K – коэффициент уменьшения радиуса окружностей на последующем уровне, Level – максимальное число уровней.

Рис. 7.6.26. – Фрактал «Окружности 1» (вариант 26).

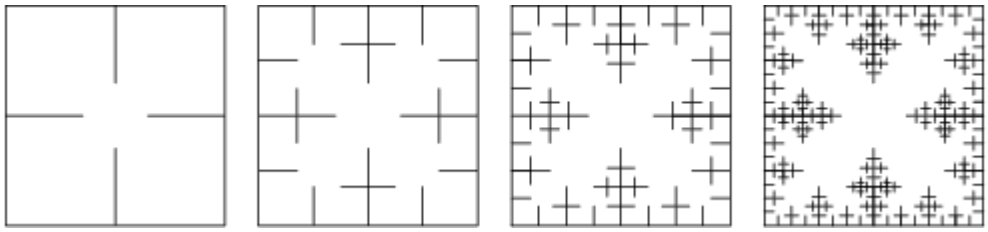


Параметры: X,Y – координаты центра фигуры, L – длина стороны треугольника, M – длина отрезков на первом уровне, K0 – коэффициент уменьшения длины отрезков на последующем уровне, K1 – коэффициент, показывающий, во сколько раз длина линий, образующих перекрестие меньше длины отрезка на текущем уровне, Level – максимальное число уровней. Этапы построения (уровни фрактала) показаны слева направо.

Рис. 7.6.27. – Фрактал «Ледяной узор 1» (вариант 27).

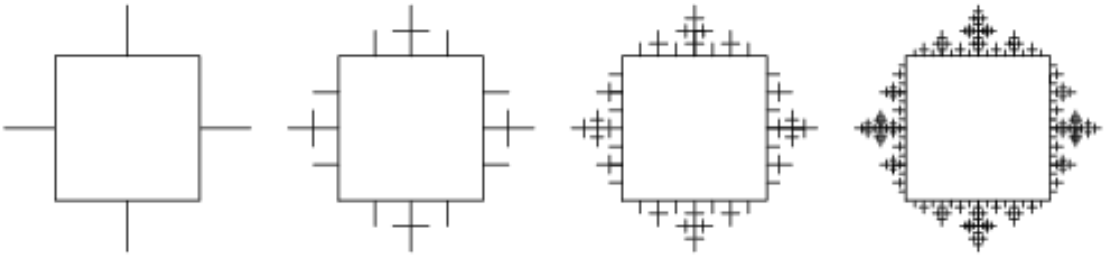
Параметры: X,Y – координаты центра фигуры, L – длина стороны треугольника, M – длина отрезков на первом уровне, K0 – коэффициент уменьшения длины отрезков на последующем уровне, K1 – коэффициент, показывающий, во сколько раз длина линий, образующих перекрестие меньше длины отрезка на текущем уровне, Level – максимальное число уровней. Этапы построения (уровни фрактала) показаны слева направо.

Рис. 7.6.28. – Фрактал «Ледяной узор 2» (вариант 28).



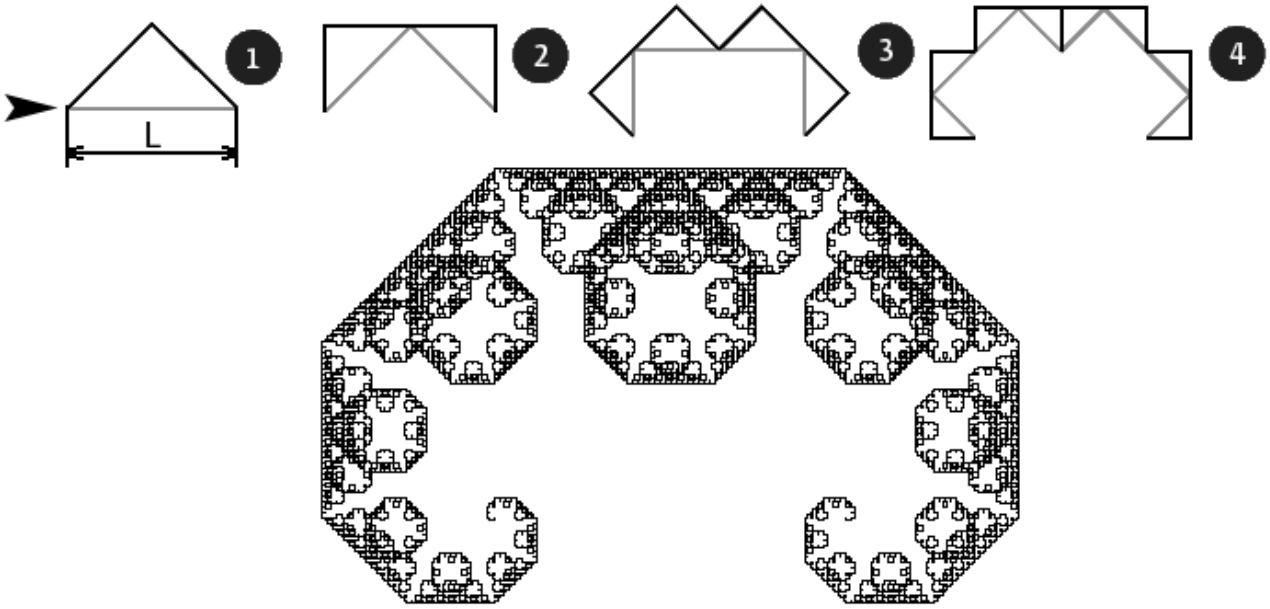
Параметры: X,Y – координаты центра фигуры, L – длина стороны квадрата, M <=L/2 – длина отрезков на первом уровне, K0 – коэффициент уменьшения длины отрезков, примыкающих к сторонам квадрата, на последующем уровне, K1 – коэффициент уменьшения длины отрезков, пересекающих отрезки, примыкающие к сторонам квадрата, Level – максимальное число уровней. Этапы построения (уровни фрактала) показаны слева направо.

Рис. 7.6.29. – Фрактал «Ледяной узор 3» (вариант 29).



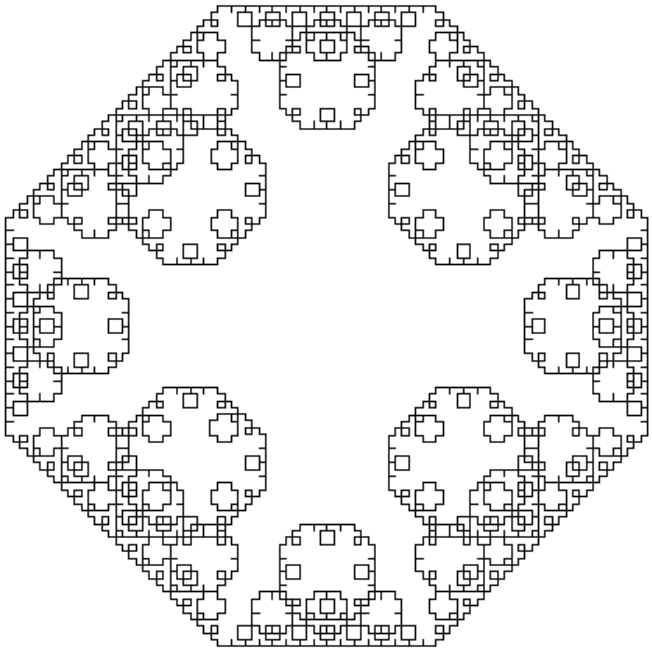
Параметры: X,Y – координаты центра фигуры, L – длина стороны квадрата, M <=L/2 – длина отрезков на первом уровне, K0 – коэффициент уменьшения длины отрезков, примыкающих к сторонам квадрата, на последующем уровне, K1 – коэффициент уменьшения длины отрезков, пересекающих отрезки, примыкающие к сторонам квадрата, Level – максимальное число уровней. Этапы построения (уровни фрактала) показаны слева направо.

Рис. 7.6.30. – Фрактал «Ледяной узор 4» (вариант 30).



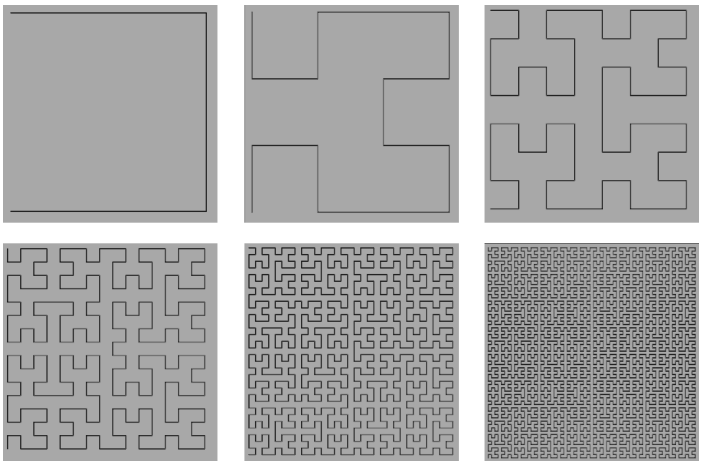
Параметры: X,Y – координаты левого нижнего угла на первом уровне построения (отмечено стрелкой), L – длина стороны треугольника на первом уровне, Level – максимальное число уровней. Этапы построения (уровни фрактала) показаны слева направо вверху. Принцип построения достаточно прост: для каждого отрезка вышестоящего уровня достраивается прямоугольный равнобедренный треугольник, так что этот отрезок выступает его гипотенузой, после чего отрезок заменяется катетами этого треугольника.

Рис. 7.6.31. – Фрактал «Кривая Леви» (вариант 31).



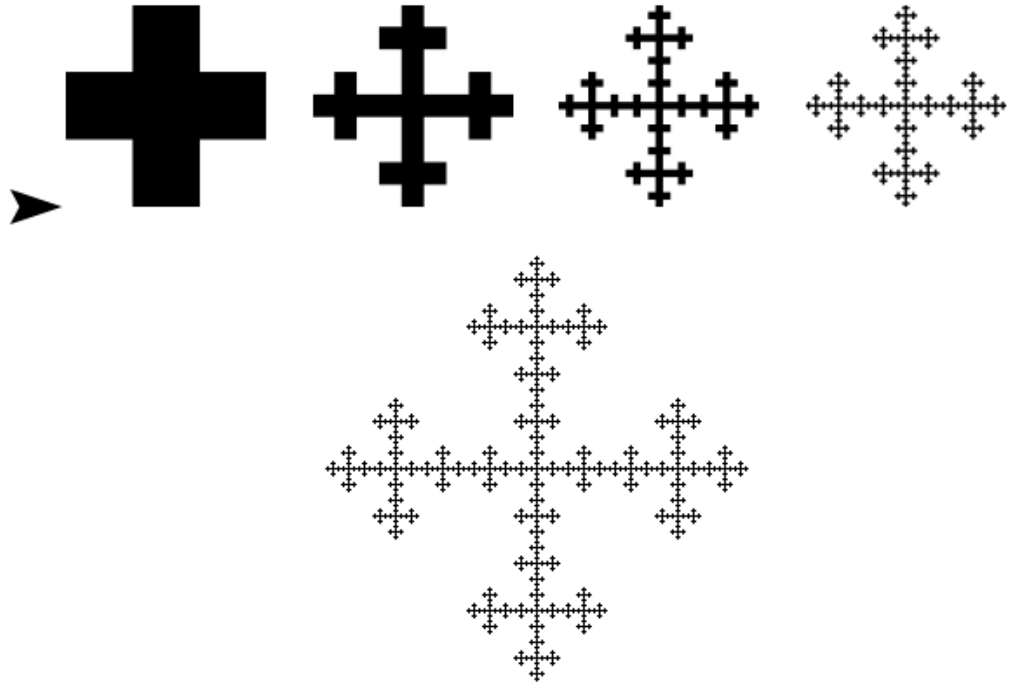
Остров Леви получается, если вместо отрезка на первом уровне взять квадрат. Принцип построения фрактала такой же как и для фрактала «кривая Леви» (см. рис. 7.6.31). Параметры: X,Y – координаты левого нижнего угла квадрата на первом уровне построения, L – длина стороны квадрата на первом уровне, Level – максимальное число уровней.

Рис. 7.6.32. – Фрактал «Остров Леви» (вариант 32).



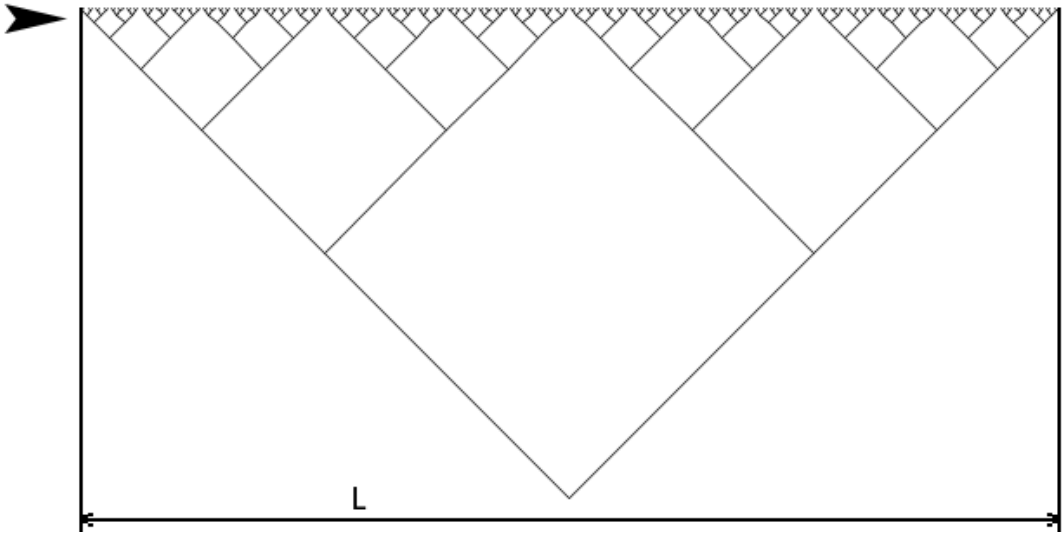
Параметры: X,Y – координаты левого нижнего угла фигуры на первом уровне построения, L – длина стороны фигуры на первом уровне, Level – максимальное число уровней. Принцип построения: каждый отрезок предыдущего уровня разбивается на три части. Далее, для первого по (ходу движения по кривой) отрезка его первая часть заменяется на П–образный элемент, для второго отрезка его вторая часть заменяется на П–образный элемент, для третьего отрезка его третья часть заменяется на П–образный элемент и т.п. Процесс для текущего уровня повторяется до тех пор, пока обход кривой не будет завершен. На рисунке показаны первые 6 уровней построения фрактала.

Рис. 7.6.33. – Фрактал «Кривая Гильберта» (вариант 33).



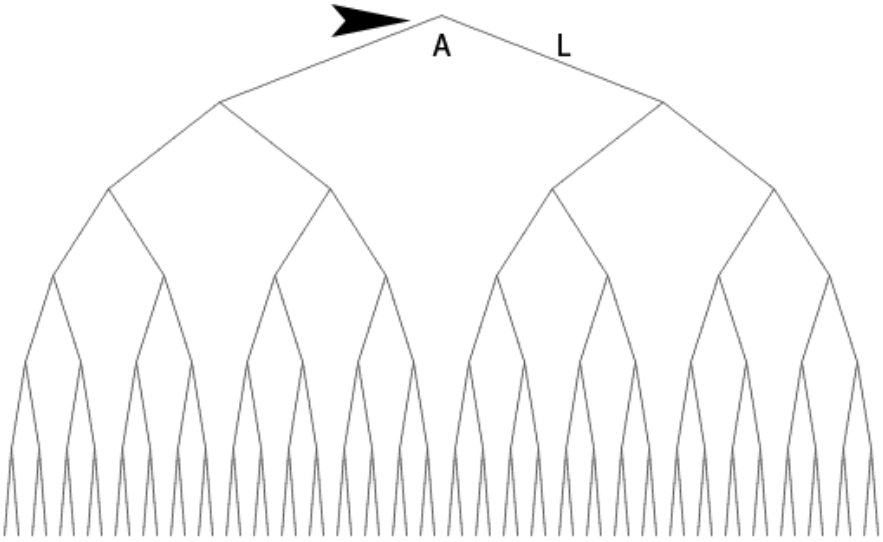
Параметры: X,Y – координаты левого нижнего угла ограничивающего квадрата для креста на первом уровне построения (отмечено стрелкой), L – длина стороны этого квадрата на первом уровне, Level – максимальное число уровней. Алгоритм построения: квадрат разбивается на 9 меньших квадратов, после чего необходимо удалить четыре угловых квадрата. К оставшимся пяти квадратам применяется аналогичная процедура на последующих уровнях.

Рис. 7.6.34. – Фрактал «Снежинка Виссека» (вариант 34).



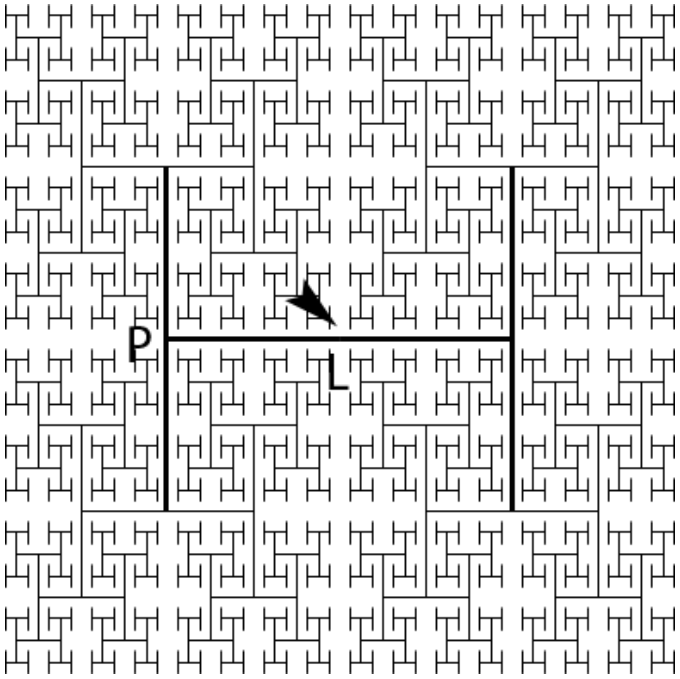
Параметры: X,Y – координаты левого верхнего угла дерева (отмечено стрелкой), L – ширина дерева, Level – максимальное число уровней.

Рис. 7.6.35. – Фрактал «V–дерево» (вариант 35).



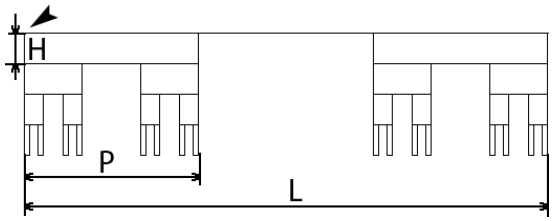
Параметры: X,Y – координаты корня дерева (отмечено стрелкой), L – длина ветви дерева на первом уровне, A – угол между ветвями на первом уровне, K0 – коэффициент уменьшения длины ветви на последующем уровне, K1 – коэффициент уменьшения угла между ветвями на последующем уровне, Level – максимальное число уровней.

Рис. 7.6.36. – Фрактал «Бинарное дерево» (вариант 36).



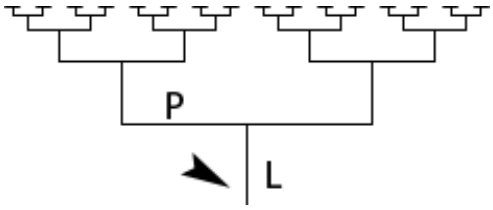
Параметры: X,Y – координаты центра перекладины буквы H (отмечено стрелкой), L – длина перекладины на первом уровне построения, P – высота вертикальных линий на первом уровне построения, K – коэффициент масштабирования размеров на последующих уровнях, Level – максимальное число уровней. Для понимания построения первый уровень показан жирной линией, на деле же все линии должны быть одинаковой толщины.

Рис. 7.6.37. – Фрактал «H–фрактал» (вариант 37).



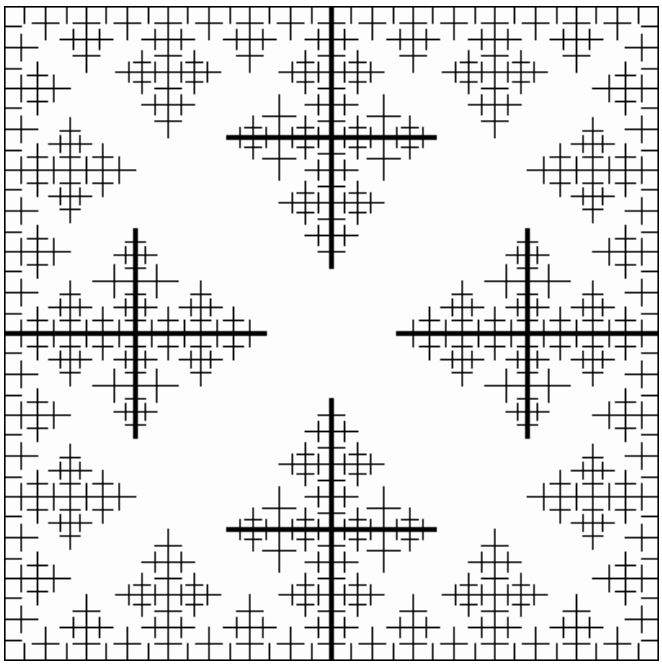
Параметры: X,Y – координаты левого верхнего угла гребня (отмечено стрелкой), H – высота зуба, L – длина гребня, K = P/L – коэффициент сужения гребенки, Level – максимальное число уровней.

Рис. 7.6.38. – Фрактал «Гребень Кантора» (вариант 38).



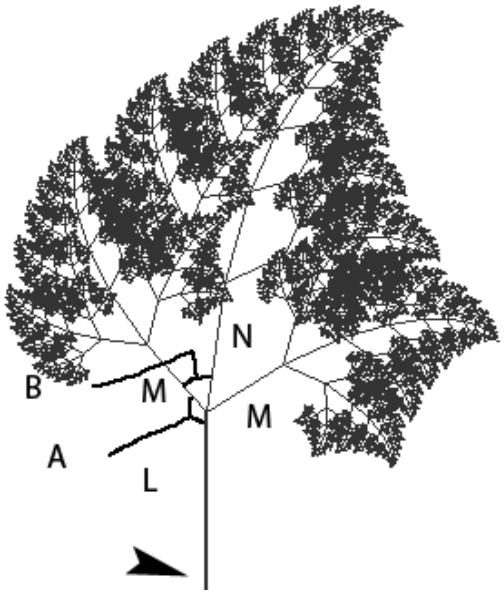
Параметры: координаты корня дерева (отмечено стрелкой), L – длина ствола дерева на первом уровне, P – длина ветви дерева на первом уровне, K0 – коэффициент уменьшения длин ствола и ветви на последующем уровне, Level – максимальное число уровней.

Рис. 7.6.39. – Фрактал «Двоичное дерево» (вариант 39).



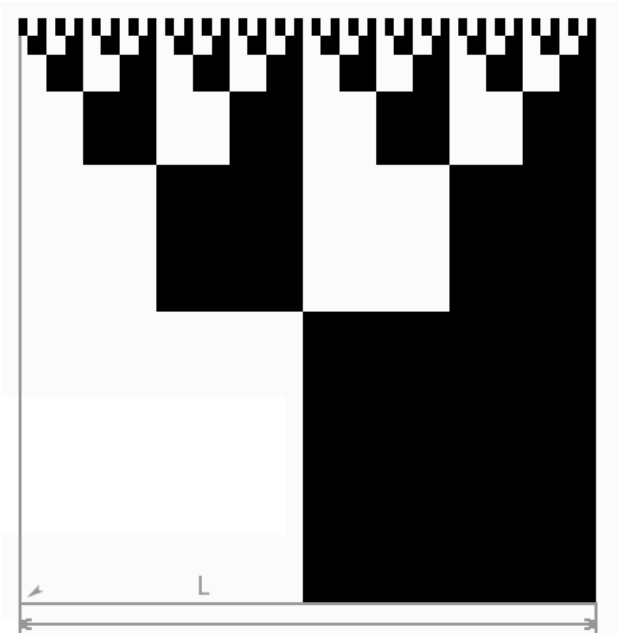
Параметры: X,Y – координаты левого нижнего угла квадрата, L – длина стороны квадрата, P,M – длины горизонтальной и вертикальной линий на первом уровне (показаны жирным), Level – максимальное число уровней. На данном рисунке образующие линии показаны жирным для понимания, на деле же все линии имеют одну толщину в 1 пиксель. Коэффициент уменьшения длин линий последующих уровней равен 0.5.

Рис. 7.6.40. – Фрактал «Ледяной квадрат» (вариант 40).



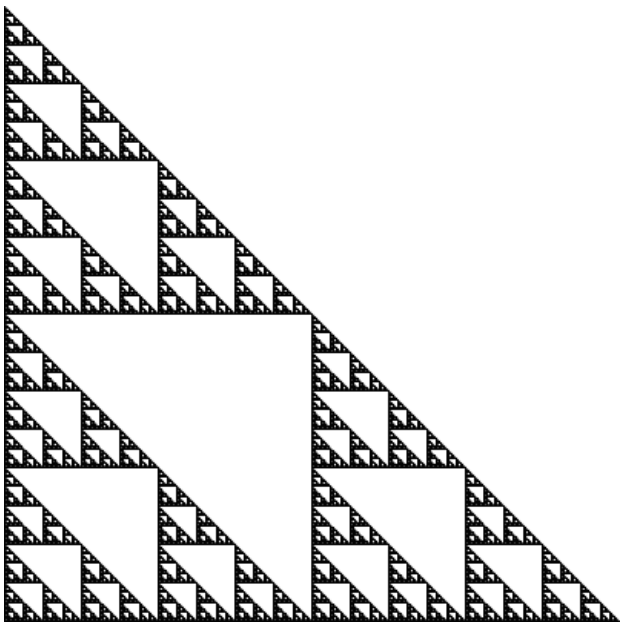
Параметры: X,Y – координаты корня папоротника (отмечено стрелкой), L – длина ствола на первом уровне, A – угол наклона первой ветви относительно ствола, B – угол между ветвями, K0 = M/L – коэффициенты уменьшения длин первой и третьей веток папоротника на последующих уровнях, K1 = N/L – коэффициент уменьшения длины второй ветки папоротника на последующих уровнях, Level – максимальное число уровней. Длины M, N показаны для понимания, параметрами не являются и вводить в программе их не нужно.

Рис. 7.6.41. – Фрактал «Папоротник» (вариант 41).



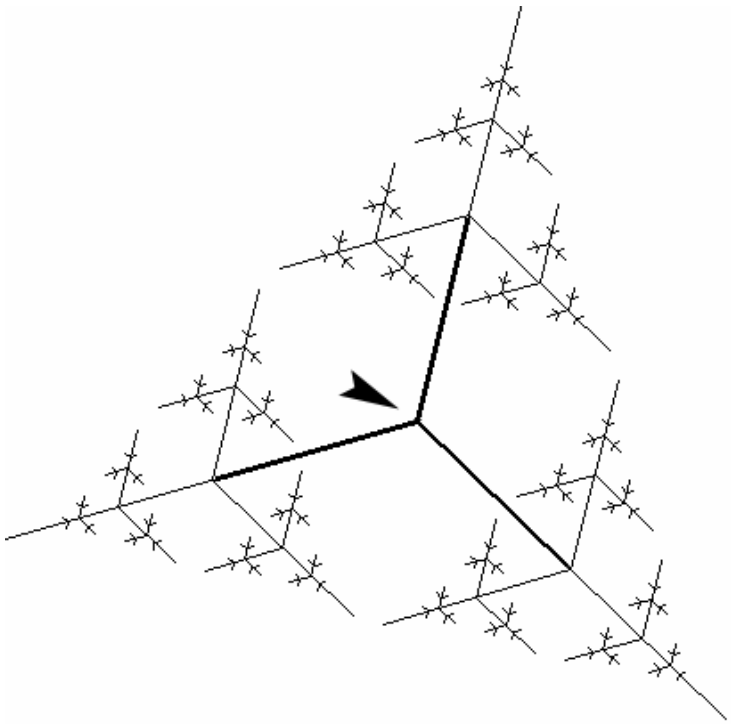
Параметры: X,Y – координаты левого нижнего угла ограничивающего квадрата для фигуры (отмечено стрелкой), L – длина стороны этого квадрата, Level – максимальное число уровней. В данном варианте все коэффициенты уменьшения длины или ширины составляют 0.5.

Рис. 7.6.42. – Фрактал «Разноцветные прямоугольники» (вариант 42).



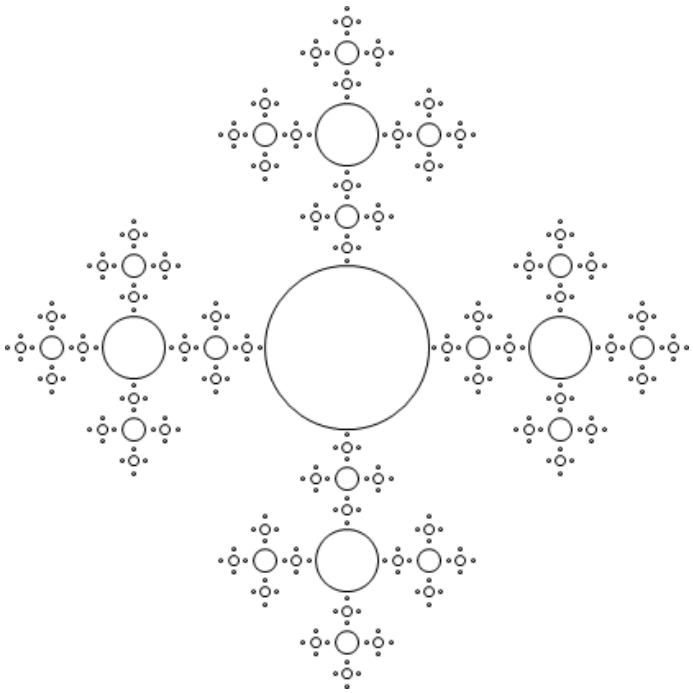
Параметры: X,Y – координаты левого нижнего угла прямоугольного треугольника, L – длина катета этого треугольника, Level – максимальное число уровней.

Рис. 7.6.43. – Фрактал «Треугольник Серпинского 2» (вариант 43).



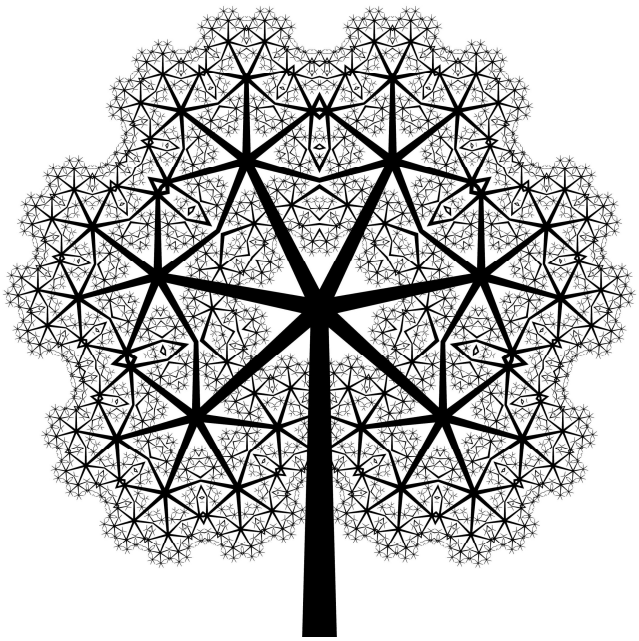
Параметры: X,Y – координаты центра дерева (отмечено стрелкой), A – угол наклона дерева относительно горизонта, L – длина ветви дерева первого уровня (отмечено жирным), K – коэффициент уменьшения длины ветви на последующих уровнях, Level – максимальное число уровней. В данном варианте задания все углы между ветвями равны 120°.

Рис. 7.6.44. – Фрактал «Троичное дерево» (вариант 44).



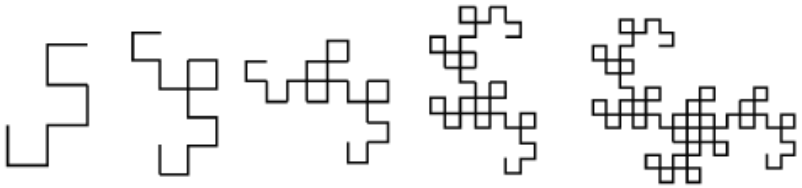
Параметры: X,Y – координаты центра фигуры, R – радиус окружности первого уровня, L – расстояние между центрами окружностей первого и второго уровней, K – коэффициент уменьшения радиуса R и расстояния L на последующих уровнях, Level – максимальное число уровней.

Рис. 7.6.45. – Фрактал «Окружности 2» (вариант 45).



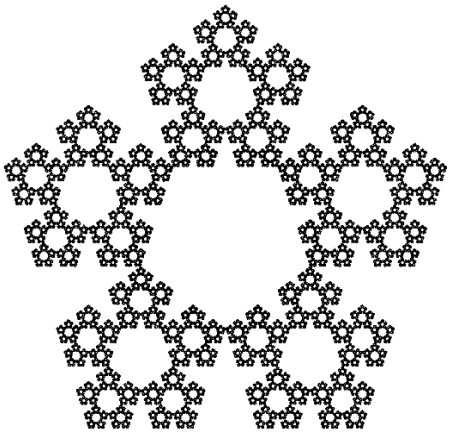
Параметры: X,Y – координаты середины ствола дерева, W – ширина ствола дерева, L – длина ствола дерева, K0 – коэффициент сужения ствола на текущем уровне, K1 – коэффициент уменьшения длин ветвей на последующих уровнях, P – число ветвей на уровне, Level – максимальное число уровней.

Рис. 7.6.46. – Фрактал «Клен 2» (вариант 46).



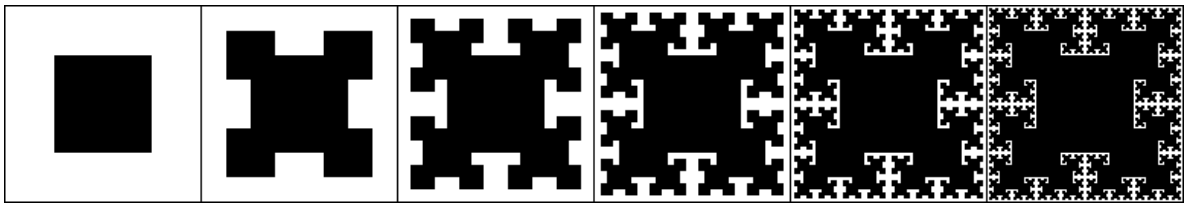
Параметры: X,Y – координаты начальной точки кривой, L – длина одного отрезка, Level – максимальное число уровней. Для построения кривой дракона необходимо создать копию всех отрезков предыдущего уровня и повернуть получившуюся фигуру на 90°. Далее следует совместить начало этой фигуры с концом предыдущей (полученной на предыдущем уровне). Этот процесс необходимо повторять рекурсивно до окончания построения.

Рис. 7.6.47. – Фрактал «Дракон Хартера–Хейтуэя» (вариант 47).



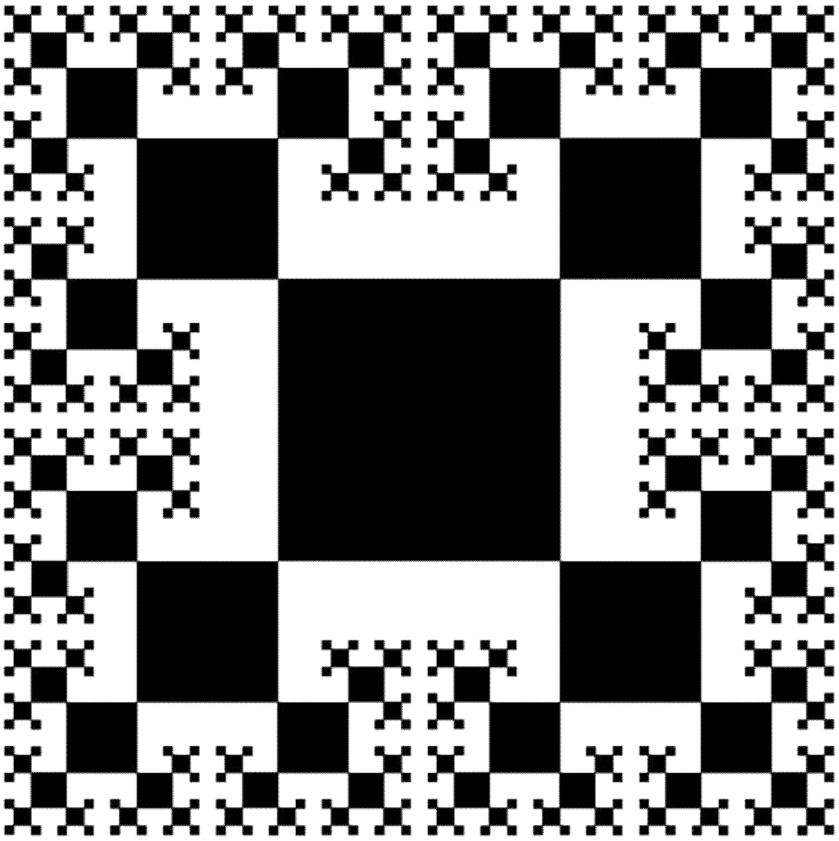
Параметры: X,Y – координаты центра фигуры, R – радиус описанной окружности пентаграммы первого уровня, Level – максимальное число уровней.

Рис. 7.6.48. – Фрактал «Звезда Дюрера» (вариант 48).



Параметры: X,Y – координаты левого нижнего угла квадрата на первом уровне построения, L – длина стороны этого квадрата, Level – максимальное число уровней. Принцип построения: сначала необходимо построить черный квадрат размером L/2, после чего в каждом выпуклом углу (после первого уровня все углы выпуклые, но так будет не всегда) необходимо построить квадрат с размерами в два раза меньшими, причем центр этих квадратов соответствует координатам выпуклых углов предыдущего уровня.

Рис. 7.6.49. – Фрактал «Т–квадрат» (вариант 49).



Параметры: X,Y – координаты левого нижнего угла квадрата первого уровня, L – длина стороны этого квадрата, Level – максимальное число уровней. Для построения фрактала сначала необходимо построить черный квадрат в центре размером L. После этого рядом с каждой вершиной квадрата строятся еще квадраты, размер которых составляет половину от его размеров. Далее процесс повторяется рекурсивно, пока не будет достигнуто нужное количество шагов.

Рис. 7.6.50. – Фрактал «Квадратная скатерть» (вариант 50).

Лабораторная работа №8
Порождающие паттерны проектирования

Цель работы: ознакомиться с принципами построения порождающих паттернов проектирования, научиться использовать их при разработке программного обеспечения.

8.1. Основные теоретические сведения

Паттернами проектирования (Design Patterns) называют решения часто встречающихся проблем в области разработки программного обеспечения. Паттерны проектирования не являются готовыми решениями, которые можно трансформировать непосредственно в код, а представляют общее описание решения проблемы, которое можно использовать в различных ситуациях.

Существуют несколько типов паттернов проектирования, каждый из которых предназначен для решения своего круга задач:

– порождающие паттерны, предназначенные для создания новых объектов в системе;

– структурные паттерны, решающие задачи компоновки системы на основе классов и объектов;

– паттерны поведения, предназначенные для распределения обязанностей между объектами в системе.

Порождающие паттерны

Пожалуй, создание новых объектов является наиболее распространенной задачей, встающей перед разработчиками программных систем. Порождающие паттерны проектирования предназначены для создания объектов, позволяя системе оставаться независимой как от самого процесса порождения, так и от типов порождаемых объектов. Сложность заключается в том, что хотя код системы и оперирует готовыми объектами через соответствующие общие интерфейсы, в процессе разработки ПО требуется создавать новые объекты, непосредственно указывая их конкретные типы. Если код их создания рассредоточен по всему приложению, то добавлять новые типы объектов или заменять существующие будет затруднительно.

В таких случаях на помощь приходит **фабрика объектов**, локализующая создание объектов. Работа фабрики объектов напоминает функционирование виртуального конструктора, – мы можем создавать объекты нужных классов, не указывая напрямую их типы. В самом простом случае, для этого используются идентификаторы типов.

Познакомившись с основными проблемами, возникающими при создании объектов новых типов, кратко рассмотрим особенности каждого из порождающих паттернов (шаблонов).

Основные виды порождающих паттернов

Паттерн **Factory Method** (фабричный метод) является самым простым паттерном для создания объектов. Он переносит создание объектов в специально предназначенные для этого классы. В его классическом варианте

вводится абстрактный класс Factory, в котором определяется интерфейс фабричного метода, а ответственность за создание объектов конкретных классов переносится на производные от Factory классы, в которых этот метод переопределяется.

Паттерн **Abstract Factory** (абстрактная фабрика) использует несколько фабричных методов и предназначен для создания целого семейства или группы взаимосвязанных объектов.

Паттерн **Builder** определяет процесс поэтапного конструирования сложного объекта, в результате которого могут получаться разные представления этого объекта.

Паттерн **Prototype** создает новые объекты с помощью прототипов. Прототип - некоторый объект, умеющий создавать по запросу копию самого себя.

Паттерн **Singleton** контролирует создание единственного экземпляра некоторого класса и предоставляет доступ к нему.

Паттерн **Object Pool** используется в случае, когда создание объекта требует больших затрат или может быть создано только ограниченное количество объектов некоторого класса.

Детальное описание работы всех видов порождающих паттернов выходит за рамки данной лабораторной работы, однако этот материал подробно рассматривается в конспекте лекций по курсу. Данная лабораторная работа посвящена созданию программного проекта с использование абстрактной фабрики, поэтому здесь ограничимся описанием только этого паттерна.

Абстрактная фабрика (Abstract Factory)

Абстрактная фабрика – порождающий паттерн проектирования, позволяющий изменять поведение системы, варьируя создаваемые объекты, при этом сохраняя интерфейсы. Он позволяет создавать целые группы взаимосвязанных объектов, которые, будучи созданными одной фабрикой, реализуют общее поведение.

Паттерн реализуется созданием абстрактного класса Factory, который представляет собой интерфейс для создания компонентов системы. Затем разрабатываются классы, реализующие этот интерфейс.

Этот паттерн предоставляет интерфейс для создания множества взаимосвязанных объектов, не специфицируя их конкретных классов. Фабрика имеет некую специализацию, создавая товары или устройства какого-либо определенного типа. Фабрика, которая выпускает, например, мебель, не может производить, например, еще и компоненты для смартфонов.

Паттерн предоставляет интерфейс для создания семейств, связанных между собой, или независимых объектов, конкретные классы которых неизвестны. От класса Factory наследуются классы конкретных фабрик, которые содержат методы создания конкретных объектов-продуктов, являющихся наследниками класса Factory, объявляющего интерфейс для их создания.

Паттерн Abstract Factory нужно использовать, если:

– система должна оставаться независимой как от процесса создания новых объектов, так и от типов порождаемых объектов. Непосредственное использование выражения new в коде приложения нежелательно;

– необходимо создавать группы или семейства взаимосвязанных объектов, исключая возможность одновременного использования объектов из разных семейств в одном контексте.

Приведем примеры групп взаимосвязанных объектов.

1. Пусть некоторое приложение с поддержкой графического интерфейса пользователя рассчитано на использование на различных платформах, при этом внешний вид этого интерфейса должен соответствовать принятому стилю для той или иной платформы. Например, если это приложение установлено на Windows-платформу, то его кнопки, меню, полосы прокрутки должны отображаться в стиле, принятом для Windows. В случае же развертывания приложения на UNIX системе его графический интерфейс должен отображаться в стиле принятом для UNIX-приложений. Получается, что функции всех элементов управления как правило будут одинаковы, хотя внешне элементы выглядят по разному. Группой взаимосвязанных объектов в этом случае будут элементы графического интерфейса пользователя для конкретной платформы.

2. В примере выполнения лабораторной работы (см. далее) речь пойдет о разработке игры-стратегии, в которой описывается военное противостояние между армиями Рима и Карфагена. Очевидно, что внешний вид, боевые порядки и характеристики для разных родов войск (пехота, лучники, конница) в каждой армии будут своими. В данном случае семейством взаимосвязанных объектов будут все виды воинов для той или иной противоборствующей стороны.

Для решения задачи по созданию семейств взаимосвязанных объектов паттерн Abstract Factory вводит понятие абстрактной фабрики. Абстрактная фабрика представляет собой некоторый абстрактный базовый класс, назначением которого является объявление интерфейсов фабричных методов, служащих для создания продуктов всех основных типов (один фабричный метод на каждый тип продукта). Производные от него классы, реализующие эти интерфейсы, предназначены для создания продуктов всех типов внутри семейства или группы. В случае нашей игры базовый класс абстрактной фабрики должен определять интерфейс фабричных методов для создания пехотинцев, лучников и конницы, а два производных от него класса будут реализовывать этот интерфейс, создавая воинов всех родов войск для той или иной армии.

Схему взаимодействия классов паттерна Abstract Factory удобно представить в виде так называемой UML-диаграммы классов. Несмотря на то, что язык UML изучается на старших курсах, диаграмму следует привести, так как она интуитивно понятна и очень наглядно описывает взаимодействие всех классов, входящих в реализацию Abstract Factory.

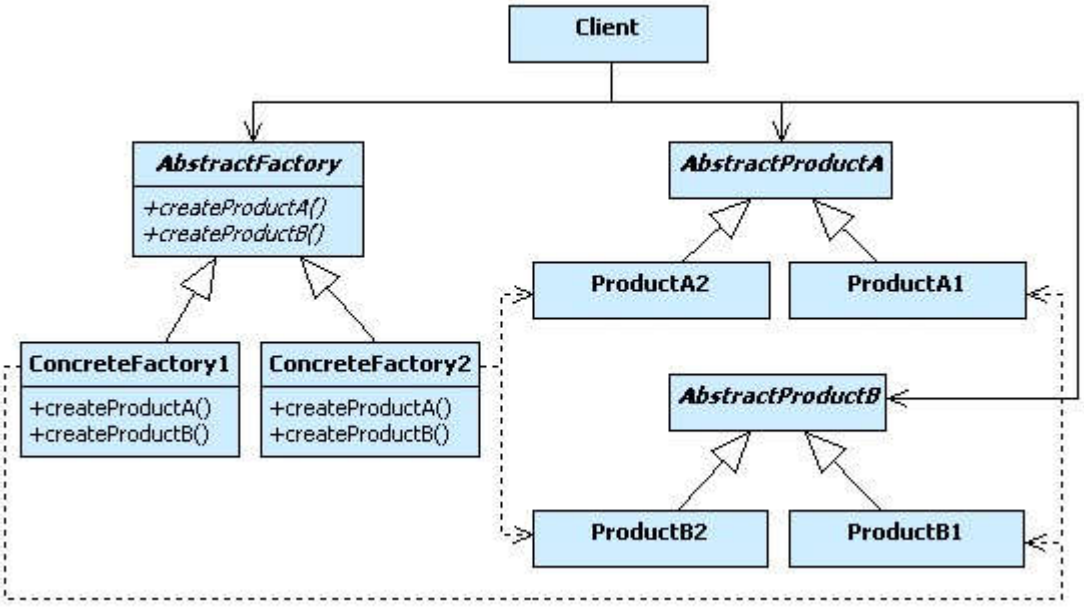


Рис. 8.1. UML-диаграмма классов паттерна Abstract Factory

На данной диаграмме прямоугольники обозначают классы, стрелки с незакрашенным концом – наследование классов, обычные стрелки – т.н. ассоциации, определяющие непрямую взаимосвязь объектов (например, наследование через промежуточные классы), штрихпунктирные стрелки – т.н. зависимости, показывающие то, что один объект зависит от другого (например, один объект выступает в роли свойства в другом объекте). Внутри прямоугольника, помимо названия класса, могут также фигурировать некоторые методы, причем публичные методы обозначаются символом «+» слева от названия, приватные – символом «-», а защищенные – символом «#».

8.2. Задание к лабораторной работе

В лабораторной работе необходимо реализовать проект, состоящий из двух групп взаимодействующих объектов. Для создания этих объектов должен использоваться паттерн Abstract Factory. В целом, приложение является консольным, которое содержит двухуровневое меню. На первом уровне пользователь выбирает группу объектов, а на втором - работу с этой группой (т.е. создание объектов группы, а также взаимодействие с объектами противоположной группы). Объекты должны создаваться в любой момент работы программы (не только при запуске), кроме того, между объектами группы должно быть реализовано взаимодействие. Перечень объектов группы и виды взаимодействий описываются в вариантах заданий. Например, имеется две типографии, которые выпускают книги, журналы, газеты (создание объектов), а также могут выкупать друг у друга продукцию и продавать ее на сторону (взаимодействие). В вариантах заданий, где создание объектов требует затрат ресурсов, необходимо обеспечить ввод начального количества ресурсов (например, бумаги, краски, клея и т.п.) или ввод некоторой денежной суммы, за

которые эти ресурсы можно купить в начале работы программы (до появления двухуровневого меню).

8.3. Пример выполнения лабораторной работы

Проект «Игровая стратегия», в которой есть две противоборствующие стороны. Каждая сторона имеет свою армию. Каждая армия – своих воинов, характеристики которых могут отличаться. Может также отличаться и требуемое количество ресурсов, необходимое для создания того или иного воина. Для простоты будем считать, что у каждого воина есть количество жизней, параметр атаки (количество жизней, отнимаемое у соперника за один ход), количество единственного ресурса «еда», необходимое для производства воина. *(Понятно, что в будущем можно будет расширять подобную стратегию – добавить новые ресурсы (например, дерево для производства лучников), добавлять показатели брони для защиты персонажей от атаки соперников и т.д., но в приводимом ниже коде этого нет).*

Файл archer.h

```
#ifndef _ARCHER_
#define _ARCHER_ 1

// Интерфейс, представляющий собой абстрактного лучника
class Archer
{
    protected:
        int totalLive;     // Количествожизнейуперсонажа
        int attackLive;    // Количество жизней, снимаемых при атаке
        int resourceCount; // Количество ресурсов для создания лучника
    public:
        // Получить количество жизней у персонажа
        virtual int getLive() = 0;
        // Получить параметр атаки
        virtual int getAttackLive() = 0;
        // Получить количество ресурсов, требуемое для создания персонажа
        virtual int getResourceCount() = 0;
        // Атаковать персонаж (метод возвращает признак жизни персонажа
        // после атаки - жив или нет, принимает атакующий параметр - сколько
        // жизней отнять при атаке)
        virtual bool attack(int attackLive) = 0;
};
#endif
```

Файл army.h

```
#include "archer.h"
#include "horseman.h"
#include "army_factory.h"
// Класс, содержащий всех воинов той или иной армии
class Army
{
    private:
        int totalArcher;       // Общее число лучников в армии
        int totalHorseman;     // Общее число кавалеристов в армии
        int totalResource;     // Общее число ресурсов в армии
```

```
        Archer** archers;       // Массив лучников в армии
        Horseman** horsemans;   // Массив кавалеристов в армии
    public:
        // Конструктор (принимает фабрику, которая будет создавать воинов,
        // количество лучников, кавалеристов, которых нужно создать, а также
        // количество ресурсов, которые есть для этого)
        // Если ресурсов недостаточно, то войны создаваться не будут
        Army(ArmyFactory* ar, int colArcher, int colHorseman, int colResource);
        // Деструктор - удаляет всех воинов
        ~Army();
        // Атакует армию неприятеля
        bool attack(Army* ar);
        // Выводит информацию о оставшихся в живых воинах армии
        void info();
        // Получает общее количество кавалеристов в армии
        int getTotalHorseman();
        // Получает общее количество лучников в армии
        int getTotalArcher();
        // Получает общее ресурсов в армии
        int getTotalResource();
};
```

Файл army_factory.h

```
#ifndef _ARMY_FACTORY_
#define _ARMY_FACTORY_ 1
#include "archer.h"
#include "horseman.h"
// Интерфейс для создания армии воинов
class ArmyFactory
{
    public:
        virtual Archer* createArcher() = 0;
        virtual Horseman* createHorseman() = 0;
};
#endif
```

Файл carthaginian_archer.h

```
#include "archer.h"
// Класс, представляющий собой лучника армии Карфагена
class CarthaginianArcher : public Archer
{
    public:
        // Конструктор
        CarthaginianArcher();
        // Получить количество жизней у персонажа
        int getLive();
        // Получить параметр атаки
        int getAttackLive();
        // Получить количество ресурсов, требуемое для создания персонажа
        int getResourceCount();
        // Атаковать персонаж (метод возвращает признак жизни персонажа после
        // атаки - жив или нет, принимает атакующий параметр - сколько жизней
        // отнять при атаке)
        bool attack(int attackLive);
};
```

Файл carthaginian_army_factory.h

```
#include "army_factory.h"
// Фабрика для создания воинов армии Карфагена
class CarthaginianArmyFactory: public ArmyFactory
{
    public:
        Archer* createArcher();
        Horseman* createHorseman();
};
```

Файл carthaginian_horseman.h

```
#include "horseman.h"
// Класс, представляющий собой кавалериста армии Карфагена
class CarthaginianHorseman : public Horseman
{
    public:
        // Конструктор
        CarthaginianHorseman();
        // Получить количество жизней у персонажа
        int getLive();
        // Получить параметр атаки
        int getAttackLive();
        // Получить количество ресурсов, требуемое для создания персонажа
        int getResourceCount();
        // Атаковать персонаж (метод возвращает признак жизни персонажа после
        // атаки - жив или нет, принимает атакующий параметр - сколько жизней
        // отнять при атаке)
        bool attack(int attackLive);
};
```

Файл horseman.h

```
#ifndef _HORSEMAN_
#define _HORSEMAN_ 1
// Интерфейс, представляющий собой абстрактного кавалериста
class Horseman
{
    protected:
        // Количество жизней у персонажа
        int totalLive;
        // Количество жизней, снимаемых при атаке
        int attackLive;
        // Количество ресурсов, необходимое для производства кавалериста
        int resourceCount;
    public:
        // Получить количество жизней у персонажа
        virtual int getLive() = 0;
        // Получить параметр атаки
        virtual int getAttackLive() = 0;
        // Получить количество ресурсов, требуемое для создания персонажа
        virtual int getResourceCount() = 0;
        // Атаковать персонаж (метод возвращает признак жизни персонажа после
        // атаки - жив или нет, принимает атакующий параметр - сколько жизней
        // отнять при атаке)
        virtual bool attack(int attackLive) = 0;
};

#endif
```

Файл roman_archer.h

```
#include "archer.h"
// Класс, представляющий собой лучника Римской армии
class RomanArcher : public Archer
{
    public:
        // Конструктор
        RomanArcher();
        // Получить количество жизней у персонажа
        int getLive();
        // Получить параметр атаки
        int getAttackLive();
        // Получить количество ресурсов, требуемое для создания персонажа
        int getResourceCount();
        // Атаковать персонаж (метод возвращает признак жизни персонажа после
        // атаки - жив или нет, принимает атакующий параметр - сколько жизней
        // отнять при атаке)
        bool attack(int attackLive);
};
```

Файл roman_army_factory.h

```
#include "army_factory.h"
// Фабрика для создания воинов Римской армии
class RomanArmyFactory: public ArmyFactory
{
    public:
        Archer* createArcher();
        Horseman* createHorseman();
};
```

Файл roman_horseman.h

```
#include "horseman.h"

// Класс, представляющий собой кавалериста Римской армии
class RomanHorseman : public Horseman
{
    public:
        // Конструктор
        RomanHorseman();
        // Получить количество жизней у персонажа
        int getLive();
        // Получить параметр атаки
        int getAttackLive();
        // Получить количество ресурсов, требуемое для создания персонажа
        int getResourceCount();
        // Атаковать персонаж (метод возвращает признак жизни персонажа после
        // атаки - жив или нет, принимает атакующий параметр - сколько жизней
        // отнять при атаке)
        bool attack(int attackLive);
};
```

Файл army.cpp

```
#include <iostream>
#include "army.h"
using namespace std;
```

```
Army::Army(ArmyFactory* ar, int colArcher, int colHorseman, int colResource)
{
    this -> totalArcher = 0;
    this -> totalHorseman = 0;
    this -> totalResource = colResource;
    this -> archers   = new Archer*[colArcher];
    this -> horsemans = new Horseman*[colHorseman];
    // Создаем лучников
    for(int i=0; i<colArcher; i++)
    {
        // Попытаться создать лучника
        Archer* a = ar->createArcher();
        // Если при этом количество потраченных ресурсов меньше доступного числа
        // ресурсов, то записываем созданного лучника в массив, иначе выходим из
        // цикла
        if (a -> getResourceCount() <= this -> totalResource)
        {
            this -> archers[this -> totalArcher] = a;
            this -> totalArcher++;
            // Уменьшаем количество ресурсов
            this -> totalResource -= a->getResourceCount();
        }
        else
            break;
    }
    // Аналогично для кавалеристов
    for(int i=0; i<colHorseman; i++)
    {
        // Попытаться создать кавалериста
        Horseman* h = ar->createHorseman();
        // Если при этом количество потраченных ресурсов меньше доступного числа
        // ресурсов, то записываем созданного лучника в массив, иначе выходим из
        // цикла
        if (h -> getResourceCount() <= this -> totalResource)
        {
            this -> horsemans[this -> totalHorseman] = h;
            this -> totalHorseman++;
            // Уменьшаем количество ресурсов
            this -> totalResource -= h->getResourceCount();
        }
        else
            break;
    }
}
int Army::getTotalHorseman()
{
    // Получает общее количество кавалеристов в армии
    return this -> totalHorseman;
}
int Army::getTotalArcher()
{
    // Получает общее количество лучников в армии
    return this -> totalArcher;
}
int Army::getTotalResource()
{
    // Получает общее ресурсов в армии
    return this -> totalResource;
}
void Army::info()
{
    // Выводит информацию о всех оставшихся в живых воинах
    cout << "Лучники:" << endl;
```

```
    for (int i = 0; i <this -> totalArcher; i++)
    {
        cout << "Лучник " << (i+1) << ": осталось ";
        cout << this -> archers[i] -> getLive() << " жизней" << endl;
    }
    cout << "Кавалеристы:" << endl;
    for (int i = 0; i <this -> totalHorseman; i++)
    {
        cout << "Кавалерист " << (i+1) << ": осталось ";
        cout << this -> horsemans[i] -> getLive() << " жизней" << endl;
    }
}
bool Army::attack(Army* ar)
{
    // Атакует армию неприятеля (ar)
    // возвращает true, если армия победила неприятельскую
    // Каждый воин текущей армии (this) выбирает случайным образом воина
    // из армии неприятеля (ar), после чего атакует его
    for (int i = 0; i <this -> totalArcher; i++)
    {
        int i1 = ar -> getTotalHorseman();
        int i2 = ar -> getTotalArcher();
        // Соперников неприятеля не осталось - армия победила
        if (i1 == 0 && i2 == 0)
            return true;
        int randType;
        int randNum;
        // Если остались только кавалеристы
        if (i1 != 0 && i2 == 0)
        {
            randType = 0;
            randNum = rand() % i1;
        }
        // Если остались только лучники
        if (i1 == 0 && i2 != 0)
        {
            randType = 1;
            randNum = rand() % i2;
        }
        // Если остались и те, и другие - тип воина выберем случайно
        if (i1 != 0 && i2 != 0)
        {
            randType = rand() % 2;
            if (randType == 0)
                randNum = rand() % i1;
            else
                randNum = rand() % i2;
        }
        // Атакуем случайно выбранного воина
        bool murdered;
        if (randType == 0)
            murdered = ar -> horsemans[randNum] -> attack(
                this -> archers[i] -> getAttackLive());
        else
            murdered = ar -> archers[randNum] -> attack(
                this -> archers[i] -> getAttackLive());
        // Если воин погиб - удалить его из массива воинов
        if (murdered == true)
        {
            if (randType == 0)
            {
                delete ar -> horsemans[randNum];
                for (int i3 = randNum; i3 < i1 - 1 ; i3++)
                    ar -> horsemans[i3] = ar -> horsemans[i3 + 1];
```

```
            ar -> totalHorseman --;
        }
        else
        {
            delete ar -> archers[randNum];
            for (int i3 = randNum; i3 < i1 - 1 ; i3++)
                ar -> archers[i3] = ar -> archers[i3 + 1];
            ar -> totalArcher --;
        }
    }
}
// Все то же самое, но для кавалериста
for (int i = 0; i <this -> totalHorseman; i++)
{
    int i1 = ar -> getTotalHorseman();
    int i2 = ar -> getTotalArcher();
    // Соперников неприятеля не осталось - армия победила
    if (i1 == 0 && i2 == 0)
        return true;
    int randType;
    int randNum;
    // Если остались только кавалеристы
    if (i1 != 0 && i2 == 0)
    {
        randType = 0;
        randNum = rand() % i1;
    }
    // Если остались только лучники
    if (i1 == 0 && i2 != 0)
    {
        randType = 1;
        randNum = rand() % i2;
    }
    // Если остались и те, и другие - тип воина выберем случайно
    if (i1 != 0 && i2 != 0)
    {
        randType = rand() % 2;
        if (randType == 0)
            randNum = rand() % i1;
        else
            randNum = rand() % i2;
    }
    // Атакуем случайно выбранного воина
    bool murdered;
    if (randType == 0)
        murdered = ar -> horsemans[randNum] -> attack(
            this -> horsemans[i] -> getAttackLive());
    else
        murdered = ar -> archers[randNum] -> attack(
            this -> horsemans[i] -> getAttackLive());
    // Если воин погиб - удалить его из массива воинов
    if (murdered == true)
    {
        if (randType == 0)
        {
            delete ar -> horsemans[randNum];
            for (int i3 = randNum; i3 < i1 - 1 ; i3++)
                ar -> horsemans[i3] = ar -> horsemans[i3 + 1];
            ar -> totalHorseman --;
        }
        else
        {
            delete ar -> archers[randNum];
            for (int i3 = randNum; i3 < i1 - 1 ; i3++)
```

```
                    ar -> archers[i3] = ar -> archers[i3 + 1];
                ar -> totalArcher --;
            }
        }
    }
    return false;
}
```

Файл carthaginian_archer.cpp

```
#include "carthaginian_archer.h"
CarthaginianArcher::CarthaginianArcher()
{
    // Текущее количество жизней лучника армии Кархагена
    this -> totalLive = 50;
    // При каждой атаке персонажи данного типа будут отнимать указанное
    // количество жизней
    this -> attackLive = 7;
    // Количество ресурсов, необходимое для производства лучника армии Кархагена
    this -> resourceCount = 24;
}
int CarthaginianArcher::getLive()
{
    return this -> totalLive;
}
int CarthaginianArcher::getAttackLive()
{
    return this -> attackLive;
}
int CarthaginianArcher::getResourceCount()
{
    return this -> resourceCount;
}
bool CarthaginianArcher::attack(int attackLive)
{
    // Нас атаковали с параметром attackLive
    // Отнимаем у персонажа переданное количество жизней
    this->totalLive -= attackLive;
    if (this->totalLive <= 0)
        return true; // Персонаж погиб
    else
        return false; // Персонаж еще живет
}
```

Файл carthaginian_army_factory.cpp

```
#include "carthaginian_archer.h"
#include "carthaginian_horseman.h"
#include "carthaginian_army_factory.h"
Archer* CarthaginianArmyFactory::createArcher()
{
    return new CarthaginianArcher();
}
Horseman* CarthaginianArmyFactory::createHorseman()
{
    return new CarthaginianHorseman();
}
```

Файл carthaginian_horseman.cpp

```
#include "carthaginian_horseman.h"

CarthaginianHorseman::CarthaginianHorseman()
{
    // Текущее количество жизней кавалериста армии Кархагена
    this -> totalLive = 120;
    // При каждой атаке персонажи данного типа будут отнимать указанное
    // количество жизней
    this -> attackLive = 15;
    // Количество ресурсов, необходимое для производства кавалериста
    // армии Кархагена
    this -> resourceCount = 35;
}
int CarthaginianHorseman::getLive()
{
    return this -> totalLive;
}
int CarthaginianHorseman::getAttackLive()
{
    return this -> attackLive;
}
int CarthaginianHorseman::getResourceCount()
{
    return this -> resourceCount;
}
bool CarthaginianHorseman::attack(int attackLive)
{
    // Нас атаковали с параметром attackLive
    // Отнимаем у персонажа переданное количество жизней
    this -> totalLive -= attackLive;
    if (this -> totalLive <= 0)
        return true; // Персонажпогиб
    else
        return false; // Персонаж еще живет
}
```

Файл main.cpp

```
#include <iostream>
#include <Windows.h>
#include "roman_army_factory.h"
#include "carthaginian_army_factory.h"
#include "army.h"
using namespace std;

// Главная функция
int main()
{
    SetConsoleOutputCP(1251);
    // Создаем Римскую армию
    int colArcher; int colHoresman; int colResource;
    cout <<"РИМСКАЯ АРМИЯ - ВВОД ДАННЫХ" << endl;
    cout <<" Число лучников: "; cin >> colArcher;
    cout <<" Число кавалеристов: "; cin >> colHoresman;
    cout <<" Количество ресурсов: "; cin >> colResource;
    // Создаем фабрику для Рима
    ArmyFactory* rimf = new RomanArmyFactory();
    // Создаем армию Рима
    Army* rim = new Army(rimf, colArcher, colHoresman, colResource);
```

```
    // Выводим параметры созданной армии
    cout << "РИМСКАЯ АРМИЯ - СОЗДАНО" << endl;
    cout << " Число лучников: " << rim ->getTotalArcher() << endl;
    cout << " Число кавалеристов: " << rim -> getTotalHorseman() << endl;
    cout << " Осталось ресурсов: " << rim -> getTotalResource() << endl;
    // *****************
    // Создаем армию Карфагена
    cout << "АРМИЯ КАРФАГЕНА - ВВОД ДАННЫХ" << endl;
    cout << " Число лучников: "; cin >> colArcher;
    cout << " Число кавалеристов: "; cin >> colHoresman;
    cout << " Количество ресурсов: "; cin >> colResource;
    // Создаем фабрику для Карфагена
    ArmyFactory* carf = new CarthaginianArmyFactory();
    // Создаем армию Карфагена
    Army* car = new Army(carf, colArcher, colHoresman, colResource);
    // Выводим параметры созданной армии
    cout << "АРМИЯ КАРФАГЕНА - СОЗДАНО" << endl;
    cout << " Число лучников: " << car ->getTotalArcher() << endl;
    cout << " Число кавалеристов: " << car -> getTotalHorseman() << endl;
    cout << " Осталось ресурсов: " << car -> getTotalResource() << endl;
    cout << endl <<"******НАЧАЛО ИГРЫ*******" << endl << endl;
    while (true)
    {
        cout << "1 - Атаковать Римскую Армию"<< endl;
        cout << "2 - Атаковать Армию Карфагена"<< endl;
        cout << "3 - Выход"<< endl;
        int command;
        cin >> command;
        if (command == 1)
        {
            if (car -> attack(rim))
                cout << "Армия Карфагена: УРА, ПОБЕДА!" << endl;
            else
            {
                cout << "Армия Карфагена: - Генерал, мы пока не победили, но";
                cout << " нанесли ощутимый урон сопернику" << endl;
                cout << "Армия Карфагена: - И какой?" << endl;
                cout << "Армия Карфагена: - Вот таблица, можете сами";
                cout << " посмотреть" << endl;
                cout << "Состав Римской армии на данный момент" << endl;
                rim -> info();
            }
        }
        if (command == 2)
        {
            if (rim -> attack(car))
                cout<<"РимскаяАрмия : УРА, ПОБЕДА!"<< endl;
            else
            {
                cout << "Римская Армия: - Полководец, мы сделали все что";
                cout << " смогли, но победить Карфагена не удалось. Но мы";
                cout << " нанесли им урон" << endl;
                cout << "Римская Армия: - И какой?" << endl;
                cout << "Римская Армия: - Вот таблица, можете сами";
                cout << " посмотреть" << endl;
                cout << "Состав Армии Карфагена на данный момент" << endl;
                car -> info();
            }
        }
        if (command == 3)
            break;
    }
}
```

Файл roman_archer.cpp

```
#include"roman_archer.h"
RomanArcher::RomanArcher()
{
    // Текущее количество жизней лучника Римской армии
    this -> totalLive = 60;
    // При каждой атаке персонажи данного типа будут отнимать указанное
    // количество жизней
    this -> attackLive = 5;
    // Количество ресурсов, необходимое для производства лучника Римской армии
    this -> resourceCount = 20;
}
int RomanArcher::getLive()
{
    return this -> totalLive;
}
int RomanArcher::getAttackLive()
{
    return this -> attackLive;
}
int RomanArcher::getResourceCount()
{
    return this -> resourceCount;
}
bool RomanArcher::attack(int attackLive)
{
    // Нас атаковали с параметром attackLive
    // Отнимаем у персонажа переданное количество жизней
    this -> totalLive -= attackLive;
    if (this -> totalLive <= 0)
        return true; // Персонаж погиб
    else
        return false; // Персонаж еще живет
}
```

Файл roman_army_factory.cpp

```
#include"roman_archer.h"
#include"roman_horseman.h"
#include"roman_army_factory.h"
Archer* RomanArmyFactory::createArcher()
{
    return new RomanArcher();
}
Horseman* RomanArmyFactory::createHorseman()
{
    return new RomanHorseman();
}
```

Файл roman_horseman.cpp

```
#include"roman_horseman.h"
RomanHorseman::RomanHorseman()
{
    // Текущее количество жизней кавалериста Римской армии
    this -> totalLive = 100;
    // При каждой атаке персонажи данного типа будут отнимать указанное
    // количество жизней
    this -> attackLive = 12;
```

```
    // Количество ресурсов, необходимое для производства кавалериста
    // Римской армии
    this -> resourceCount = 30;
}
int RomanHorseman::getLive()
{
    return this -> totalLive;
}
int RomanHorseman::getAttackLive()
{
    return this -> attackLive;
}
int RomanHorseman::getResourceCount()
{
    return this -> resourceCount;
}
bool RomanHorseman::attack(int attackLive)
{
    // Нас атаковали с параметром attackLive
    // Отнимаем у персонажа переданное количество жизней
    this -> totalLive -= attackLive;
    if (this -> totalLive <= 0)
        return true; // Персонаж погиб
    else
        return false;  // Персонаж еще живет
}
```

8.4. Содержание отчета

1. Титульный лист.
2. Условие лабораторной работы.
3. UML-диаграмма классов разработанного проекта.
4. Текст программы.
5. Экранные формы с примерами работы программы.

8.5. Контрольные вопросы

1. Что такое паттерны проектирования?
2. Какие существуют типы паттернов проектирования?
3. Какие паттерны относятся к порождающим?
4. Дайте краткое описание паттерна Abstract Factory.
5. Что показывает UML-диаграмма классов?

8.6. Варианты заданий

1. Проект “Заводы по производству ноутбуков”. В проекте должна быть реализована возможность создавать ноутбуки различных типов на разных заводах. При каждом создании ноутбука тратятся некоторые ресурсы (металл, пластик). Готовый ноутбук можно продать, вернув средства, затраченные на его создание.
2. Проект “Студии кинофильмов”. В проекте должна быть реализована возможность создавать фильмы различных жанров на разных

студиях. При каждом создании фильма тратятся некоторые ресурсы (деньги). Студии могут продавать друг другу фильмы, возвращая таким образом деньги, затраченные на его создание.

3. Проект “Заводы по производству шоколадок”. В проекте должна быть реализована возможность создавать шоколадки различных типов на разных заводах. При каждом создании шоколадки тратятся некоторые ресурсы (сахар, какао). Готовую шоколадку можно продать и вернуть ресурсы, затраченные на её создание, либо использовать как ингредиент (тогда ресурсы не возвращаются).

4. Проект “Танки”. В проекте должна быть реализована возможность создавать танки различных типов на разных заводах. При каждом создании танка тратятся некоторые внутриигровые ресурсы, которые можно купить. Готовый танк можно кинуть в бой, где у него будут отниматься жизни. После боя танк можно починить.

5. Проект “Заводы по производству роботов”. В проекте должна быть реализована возможность создавать роботов различных типов на разных заводах. При создании робота тратятся некоторые ресурсы (металл, пластик). Готовые армию роботов (собираются по типу) могут воевать друг с другом, пока у одной из армий не останется живых роботов.

6. Проект “Книжные издательства”. В проекте должна быть реализована возможность печатать книги различных жанров в различных издательствах. При печати книги тратятся некоторые ресурсы (бумага, чернила). Готовую книгу можно продать в книжный магазин, библиотеку или в частные руки, вернув ресурсы, затраченные на её создание.

7. Проект “Гейм-индустрия”. В проекте должна быть реализована возможность создавать игры различных жанров на разных студиях. При каждом создании игры тратятся некоторые ресурсы (деньги). Готовую игру можно выпустить на рынок и продавать, возвращая процент суммы, затраченной на её создание.

8. Проект “Фабрика по производству планет”. В проекте должна быть реализована возможность создавать планеты различных типов в разных вселенных. При каждом создании вселенной тратятся некоторые ресурсы (радиоактивные элементы, хим.элементы, полезные ископаемые). Готовая планета обладает определённым количеством кораблем. Корабли можно пересылать между планетами разных вселенных – с условием, что на принимающую планету доходит кораблей меньше, чем было отправлено.

9. Проект “Заводы по производству автомобилей”. В проекте должна быть реализована возможность создавать автомобили различных типов на разных заводах. При каждом создании автомобиля тратятся некоторые ресурсы (металл, пластик), которые можно купить. Готовый автомобиль можно продать, вернув деньги, затраченные на ресурсы для его создания.

10. Проект “Заводы по производству скворечников”. В проекте должна быть реализована возможность создавать скворечники различных типов на разных заводах. При каждом создании скворечники тратятся некоторые ресурсы (древесина, смола, солома). Готовый скворечник можно продать,

вернув ресурсы, затраченные на его создание, или повесить на дерево(тогда ресурсы не возвращаются).

11. Проект “Ювелирные мастерские”. В проекте должна быть реализована возможность создавать ювелирные изделия различных типов на разных заводах. При каждом создании ювелирного изделия тратятся некоторые ресурсы (драгоценный металл, драгоценные камни), которые можно купить. Готовое ювелирное изделие можно продать в ювелирном салоне, вернув деньги на ресурсы.

12. Проект “Заводы по производству CD-дисков”. В проекте должна быть реализована возможность создавать диски различных типов на разных заводах. При каждом создании диска тратятся некоторые ресурсы (пластик, краска). С одного готового диска на другой можно записать информацию.

13. Проект “Лаборатории по созданию бактерий”. В проекте должна быть реализована возможность создавать бактерии различных типов в разных лабораториях. При каждом создании бактерии тратятся некоторые ресурсы (вода, колбочки). Бактерии из различных лабораторий можно смешивать, исследуя их тип деятельности: защитный, поражающий, пассивный (результат выбирается в зависимости от взаимодействующих типов).

14. Проект “Тату-салоны”. В проекте должна быть реализована возможность набить татуировки различных дизайнов в разных салонах. При набивании татуировки тратятся некоторые ресурсы (краска, бумага, бинты), которые можно купить. Различные салоны могут выкупать друг у друга татуировки различных дизайнов, выплачивая столько же денег, сколько будут стоить ресурсы на её создание.

15. Проект “Оружейные мастерские”. В проекте должна быть реализована возможность создавать оружие различных типов в различных мастерских. При создании новой единицы оружия тратятся некоторые ресурсы (металл, древесина, ткань). Готовое оружие можно опробовать в бою с оружием из другой мастерской. Оружия разного типа наносят n урона другому оружию (также в зависимости от типа). Если нанесён критический урон, оружие выходит из строя.

16. Проект “Фруктовые теплицы”. В проекте должна быть реализована возможность выращивать фрукты различных видов на разных теплицах.На выращивание одного вида фруктов тратятся некоторые ресурсы (земля, удобрение, семена, вода), которые можно купить. Теплицы могут выкупать друг у друга фрукты.

17. Проект “Заводы по производству кроссовок”. В проекте должна быть реализована возможность создавать кроссовки различных моделей на разных заводах. При каждом создании пары кроссовок тратятся некоторые ресурсы (нитки, ткань, резина), которые можно купить. Заводы могут продавать друг другу разные партии кроссовок.

18. Проект “Лаборатории по производству зелий”. В проекте должна быть реализована возможность создавать зелья различных типов в разных лабораториях. При каждом создании зелья тратятся некоторые ресурсы (вода, химикаты, растения), которые можно купить. Готовые зелья из разных

лабораторий можно смешать, получив в результате зелье с сильными/средними/слабыми побочными эффектами или без таковых, в зависимости от типов смешивающихся зелий.

19. Проект “Художественные мастерские”. В проекте должна быть реализована возможность создавать картины различных типов (масляные, акварельные, грифельные, т.п.)в разных мастерских. При каждом создании картины тратятся некоторые ресурсы (краски, холсты, грифели), которые можно купить. Готовую картину можно продать в другую мастерскую и окупить работу, либо подарить её другой мастерской (тогда деньги за ресурсы не возвращаются).

20. Проект “Пекарни”. В проекте должна быть реализована возможность создавать выпечные изделия различных типов в разных пекарнях. При каждом создании одного выпечного изделия тратятся некоторые ресурсы (мука, яйца, сахар), которые можно купить. Готовые выпечные изделия можно украсить кремом, съесть или продать другой пекарне.

21. Проект “Заводы по производству красок”. В проекте должна быть реализована возможность создавать краски различных цветов на разных заводах (7 цветов радуги + чёрный и белый). При каждом создании краски тратятся некоторые ресурсы (химикаты, красители), которые можно купить. Готовые краски с различных заводов можно смешивать между собой, чтобы получить новый цвета (белый+цвет даёт приставку «светло-», чёрный+цвет = «тёмно-», помимо этого создать 4 возможных комбинации).

22. Проект “Заводы по производству настольных игр”. В проекте должна быть реализована возможность создавать настольные различных типов (семейные, кампании, текстовые, т.д.) на разных заводах. При каждом создании настольной игры тратятся некоторые ресурсы (картон, бумага, краски). Заводы могут реализовывать взаимных обмениваться настольными играми.

23. Проект “Питомники для котиков”. В проекте должна быть реализована возможность создавать котиков различных типов в разных питомниках. При каждом создании котика для его дальнейшей жизни тратятся некоторые ресурсы (корм, вода), которые можно купить. Питомники могут реализовывать взаимный обмен котиками, или котика можно передать на попечение в другие руки (тогда котик больше не числится как объект питомника).

24. Проект “Фабрики по вязанию свитеров”. В проекте должна быть реализована возможность создавать свитера различных типов на разных заводах. При каждом создании свитера тратятся некоторые ресурсы (нитки, пуговицы, спицы), которые можно купить. Фабрики могут реализовывать взаимный обмен свитерами.

25. Проект “Заводы по производству космолётов”. В проекте должна быть реализована возможность создавать космолёты различных типов (ближний/средний/дальний по максимальной дальности полёта) на разных заводах. При каждом создании космолёта тратятся некоторые ресурсы (металл, пластик), которые можно купить. Готовый космолёт можно перемещать на другие заводы.

26. Проект “Парфюмерные лаборатории”. В проекте должна быть реализована возможность создавать парфюмы различных запахов на разных лабораториях. При каждом создании парфюма тратятся некоторые ресурсы (спирт, ароматизирующие вещества), которые можно купить. Готовые парфюмы можно смешать, чтобы получить новый запах.

27. Проект «Завод по производству мягких игрушек». В проекте должна быть реализована возможность создавать игрушки разных типов на разных заводах. При каждом создании игрушки тратятся некоторые ресурсы (плюш, ткань), которые можно купить. Готовую продукцию можно продать, вернув деньги, затраченные на ресурсы для её создания.

28. Проект «Теплица по выращиванию цветов». В проекте должна быть реализована возможность выращивать цветы разных типов в разных теплицах. При выращивании растения тратятся некоторые ресурсы (удобрение, вода). Теплицы могут обмениваться семенами, которые получаются с цветов (с каждого типа – разное количество семян). Когда семена собирают, цветок уже не числится в списках теплицы.

29. Проект «Заводы по производству стекла». В проекте должна быть реализована возможность создавать стекла различных видов на разных заводах. При каждом создании стекла тратятся некоторые ресурсы (песок, сода). Заводы могут отсылать друг другу партии стека (в одной партии – 10 штук).

30. Проект «Заводы по производству мячей». В проекте должна быть реализована возможность создавать мячи различных видов на разных заводах. При создании мяча тратятся некоторые ресурсы (резина, краска). Заводы могут отсылать друг другу партии мячей (в одной партии – 7 штук).

31. Проект «Заводы по производству фарфоровых статуэток». В проекте должна быть реализована возможность создавать статуэтки различных видов на разных заводах. При создании статуэтки тратятся некоторые ресурсы (глина, краска), которые можно купить. Готовую продукцию можно продать.

32. Проект «Верфи по производству кораблей». В проекте должна быть реализована возможность создавать корабли различных видов на разных верфях. При создании корабля тратятся некоторые ресурсы (металл, дерево). Готовый корабль можно отправить на другую верфь.

33. Проект «Заводы по производству одежды». В проекте должна быть реализована возможность шить одежду различных видов на разных заводах. При создании одной единицы одежды тратятся некоторые ресурсы (ткань, нитки), которые можно купить. Партию готовой продукции (10 штук) можно продать, вернув процент денег за ресурсы, затраченные на её создание.

34. Проект «Заводы по производству светильников». В проекте должна быть реализована возможность создавать светильники различных видов на разных заводах. При создании светильника тратятся некоторые ресурсы (лампы, провода), которые можно купить. Готовую продукцию можно послать на другой завод, при этом продукция изнашивается. Светильник списывается со счетов, если его уровень изношенности становится равен нулю.

35. Проект «Издательства по производству почтовых марок». В проекте должна быть реализована возможность создавать марки различных

видов на разных издательствах. При создании марки тратятся некоторые ресурсы (бумага, краска). Заводы могут обмениваться марками, при этом на принимающий завод приходит на 3 марки меньше, чем было отослано.

36. Проект «Заводы по производству мороженного». В проекте должна быть реализована возможность создавать мороженное различного вида на разных заводах. При создании мороженного тратятся некоторые ресурсы (пищевые красители, добавки), которые можно купить. Готовую продукцию можно продать, вернув деньги за ресурсы, либо съесть (тогда деньги не возвращаются).

37. Проект «Заводы по производству бытовой химии». В проекте должна быть реализована возможность создавать бытовую химию (порошок, шампунь, гель для душа, т.п.) различного вида на разных заводах. При создании одной единицы продукции тратятся некоторые ресурсы (химические элементы). Готовую продукцию с разных заводов можно смешивать, получая другой продукт или испорченную смесь (результат зависит от смешиваемых типов).

38. Проект «Заводы по производству кондиционеров». В проекте должна быть реализована возможность изготовлять кондиционеры различных видов на разных заводах. При изготовлении кондиционера тратятся некоторые ресурсы (трубки, пластик), которые можно купить. Готовую продукцию можно продать другому заводу, ввернув процент денег, затраченных на производство одного изделия.

39. Проект «Заводы по производству искусственных деревьев». В проекте должна быть реализована возможность создавать деревья различных видов на разных заводах. При каждом создании дерева тратятся некоторые ресурсы (полимер, пластик). Готовые партии деревьев (10 штук) можно пересылать на другой завод.

40. Проект «Фермы по выращиванию кофейного дерева». В проекте должна быть реализована возможность выращивания кофейных деревьев различных видов на разных фермах. При каждом выращивании кофейного дерева тратятся некоторые ресурсы (вода, удобрения), которые можно купить. Зерна с готового дерева можно собрать и переслать на другую ферму. У каждого типа своё количество зёрен.

41. Проект «Кофейни». В проекте должна быть реализована возможность варить кофе различных видов в разных кофейнях. При приготовлении кофе тратятся некоторые ресурсы (зерна, сахар). Кофейни могут обмениваться партиями кофе, в партии должен быть хотя бы один кофе каждого типа.

42. Проект «Строительные компании». В проекте должна быть реализована возможность создавать здания различных конструкций в разных компаниях. При строительстве здания тратятся некоторые ресурсы (кирпич, бетон). Компании могут обмениваться проектами конструкций, однако нужно предусмотреть, например, что один проект одного типа может обмениваться только на два проекта другого типа.

43. Проект «Компании по разработке мобильных приложений». В проекте должна быть реализована возможность разрабатывать приложения для различных отраслей деятельности в разных компаниях. При разработке приложения тратятся некоторые ресурсы (деньги). Компании могут выкупать друг у друга проекты (готовый проект стоит дороже, чем сумма, за которую он создавался).

44. Проект «Туристические агентства». В проекте должна быть реализована возможность создавать туры в различных направлениях в разных агентствах. При планировании тура тратятся некоторые ресурсы (деньги). Готовый тур можно продать клиенты за ту же сумму, которая была затрачена на его создание, или продать в другое турагентство с наценкой.

45. Проект «Рекламные агентства». В проекте должна быть реализована возможность создавать рекламу различных типов в разных агентствах. При создании рекламы тратятся некоторые ресурсы (фотобумага, кинопленка), которые можно купить. Рекламные агентства могут выкупать друг у друга проекты (готовый проект стоит дороже, чем затраты на его создание).

46. Проект «Студии звукозаписи». В проекте должна быть реализована возможность создавать звуковые записи различных типов в разных студиях. При создании звуковых записей тратятся некоторые ресурсы (микрофон, самплер). Готовую звукозапись можно записать на диск через другую студию звукозаписи. Звукозапись, которая уже была записана, нельзя перезаписать в другой студии.

47. Проект «Глиняные мастерские». В проекте должна быть реализована возможность создавать глиняные изделия различных видов в разных мастерских. При создании изделия тратятся некоторые ресурсы (глина, гончарный круг). Готовое изделие можно отправить в другую мастерскую, при этом изделие изнашивается. Изделие со степенью изношенности 0 больше использовать нельзя.

48. Проект «Заводы по переработке мусора». В проекте должна быть реализована возможность перерабатывать мусор различных типа на разных заводах. При переработке мусора тратятся некоторые ресурсы (машины, модули сортировки). В результате переработки мусора получается некоторое количество материала (в соответствии с типом мусора), который можно пересылать между заводами.

49. Проект «Заводы по производству накладных ресниц». В проекте должна быть реализована возможность создавать ресницы различных видов на разных заводах. При создании изделия тратятся некоторые ресурсы (синтетика, шелк), которые можно купить. Готовое изделие можно наклеить на свои ресницы или продать, вернув деньги, отданные за ресурсы для их создания.

50. Проект «Майнеры криптовалюты». В проекте должна быть реализована возможность создавать объекты криптовалюты различных видов в разных майнерских компаниях. При создании изделия тратятся некоторые ресурсы (электричество, деньги). Готовые изделия можно обменивать по курсу между майнерскими компаниями. Каждому виду криптовалюты соответствует свой обменный курс по отношению к другим криптовалютам.

Заключение

В методических указаниях приводятся методические рекомендации по дисциплине «Объектно-ориентированное программирование» для студентов, обучающихся по направлениям подготовки 02.03.01 «Математика и компьютерные науки», 09.03.02 «Информационные системы и технологии» всех форм обучения.

Методические указания включают 3 лабораторные работы, выполнение которых рассчитано на закрепление теоретического материала и получение практических навыков программирования на языке C++ по основным темам изучаемой дисциплины. В настоящих указаниях приводятся рекомендации для изучения второй, заключительной, части курса.

Список рекомендованных источников

1. Павловская Т. А. C++. Объектно-ориентированное программирование: Практикум [Текст] / Т. А. Павловская, Ю. А. Щупак // СПб.: Питер, 2006. — 265 с: ил.
2. Шилдт Г. C++: руководство для начинающих [Текст] / Герберт Шилдт // Пер. с англ. — М.: Изд. дом «Вильямс», 2005. — 2-е издание. — 672 с. : ил. — Парал. тит. англ.
3. Шилдт Г. Искусство программирования на C++ [Текст] / Герберт Шилдт // СПб.: БХВ Петербург, 2005. — 496 с. : ил.
4. Лафоре Р. Объектно-ориентированное программирование в C++. Классика Computer Science [Текст] / Роберт Лафоре // СПб.: Питер, 2022. — 4-е издание. — 928 с.: ил. — (Серия «Классика computer science»).
5. Фримен Э. Head First: Паттерны проектирования [Текст] / Эрик Фримен, Элизабет Робсон // Пер. с англ. — СПб.: Питер, 2018. —656 с.: ил.
6. TX Library Документация [Электронный ресурс] / сост. Илья Дединский // URL: http://storage.ded32.net.ru/Lib/TX/TXUpdate/Doc/HTML.ru/ (дата обращения 01.02.2022).

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к выполнению лабораторных работ
по дисциплине «Объектно-ориентированное программирование»
(часть 2)

Составитель:

Павлий Виталий Александрович – кандидат технических наук, доцент кафедры компьютерного моделирования и дизайна ГОУВПО «ДОННТУ»

Ответственный за выпуск:

Карабчевский Виталий Владиславович – кандидат технических наук, доцент, заведующий кафедрой компьютерного моделирования и дизайна ГОУВПО «ДОННТУ»