МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ "ВИТЕБСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ П.М. МАШЕРОВА"
Факультет математический
Кафедра информатики и информационных технологий
КУРСОВАЯ РАБОТА
по дисциплине Технология программирования и методы алгоритмизации
Графика в PascalABC
Витебск, 2013
Содержание

Введение

В данной курсовой работе рассматривается тема "Графика в PascalABC". Тема курсовой работы выбрана не случайно, так как графика является довольно интересной областью программирования. Данная курсовая работа показывает, что можно создавать простые рисунки не только в графических программах, таких как Paint, Adobe Photoshop, Corel Draw, но и в среде языка программирования PascalABC. Целью курсовой работы является разработка графического проекта в среде программирования PascalABC. Графические возможности PascalABC будут реализованы на примере изображения фрактального дерева.
Задачи: 1)Познакомиться с историей становления языка программирования PascalABC.
2)Изучить основные понятия графики, функции и процедуры работы с графикой PascalABC.

3) Освоить навыки создания графических изображений в среде программирования PascalABC
4)Создать графическое изображение в среде программирования PascalABC с использованием фракталов.
Курсовая работа состоит из двух основных разделов:
1. Аналитическая часть;
2. Создание графического проекта.
В первом разделе описывается история языка PascalABC, основные понятия графики, работа с графикой в PascalABC, основные команды построения изображения, графические модули, дается основная структура программы. Во втором разделе описываются этапы создания фрактального дерева, дается краткое изложение основных элементов рисунка.
Текст созданной программы прикладываются в данной курсовой работе в разделе "Приложение".
1. Теоретическая часть

1.1 Краткая история становления языка программирования Pascal

Pascal ABC разработан в 2002 году сотрудниками факультета математики, механики и компьютерных наук Южного федерального университета (Ростов-на-Дону, Россия) во главе с С.С. Михалковичем. Целью авторов было создание обучающей среды программирования, более современной, чем Borland Pascal и Turbo Pascal, более простой для изучения, чем Borland Delphi, но в то же время близкой к стандартным компиляторам языка.
Система Pascal ABC основана на языке Delphi Pascal и призвана осуществить постепенный переход от простейших программ к модульному, объектно-ориентированному, событийному и компонентному программированию. Некоторые языковые конструкции в Pascal ABC допускают, наряду с основным, упрощенное использование, что позволяет использовать их на ранних этапах обучения. Например, в модулях может отсутствовать разделение на секцию интерфейса и секцию реализации. В этом случае модули устроены практически так же, как и основная программа, что позволяет приступить к их изучению параллельно с темой "Процедуры и функции". Тела методов можно определять непосредственно внутри классов (в стиле Java и C#), что позволяет создавать классы практически сразу после изучения записей, процедур и функций.
Ряд модулей системы программирования Pascal ABC специально создавался для учебных целей:
· Модуль растровой графики GraphABC обходится без объектов, хотя его возможности практически совпадают с графическими возможностями Borland Delphi. Он доступен в несобытийных программах и позволяет легко создавать анимацию без мерцания.
· Модуль Events позволяет создавать простейшие событийные программы без использования объектов (события представляют собой обычные процедурные

переменные).
· Модули Timers и Sounds позволяют создавать таймеры и звуки, которые также реализованы в процедурном стиле. Эти модули можно использовать даже в консольных программах.
· Модуль контейнерных классов Containers позволяет работать с основными структурами данных (динамические массивы, стеки, очереди, множества), реализованными в виде классов.
· Модуль векторной графики ABCObjects предназначен для быстрого изучения основ объектно-ориентированного программирования, а также позволяет создавать достаточно сложные игровые и обучающие программы.
· Модуль визуальных компонентов VCL позволяет создавать событийные приложения с главной формой в стиле Delphi. Классы VCL немного упрощены по сравнению с аналогичными классами Delphi. Имеется редактор форм и инспектор объектов. Технология восстановления формы по коду программы позволяет обойтись для приложения с главной формой одним файлом (!).
В языке Pascal ABC имеются арифметические операции с типизированными указателями (в стиле языка C), а также тип complex, предназначенный для работы с комплексными числами.
Компилятор Pascal ABC является компилятором переднего плана (front-end). Это означает, что он не генерирует исполняемый код в виде .exe-файла, а создает в результате компиляции дерево программы в памяти, которое затем выполняется с помощью встроенного интерпретатора. В итоге скорость работы программы примерно в 20 раз медленнее скорости работы этой же программы, откомпилированной в среде Borland Pascal, и в 50 раз медленнее этой программы, откомпилированной в среде Borland Delphi.
В системе Pascal ABC ученик может выполнять так называемые проверяемые задания, обеспечивающие постановку задачи со случайными исходными данными, контроль операций ввода-вывода, проверку правильности решения, а также ведение протокола решения задач.
1.2 Основные понятия графики

Экраны цветных мониторов состоят из прямоугольной решётки точек(пикселей),светящихся разным цветом. Каждый цветной пиксель образован тремя более мелкими по площади участками красного, зелёного и синего цветов. При свечении этих участков с разной интенсивностью цвета смешиваются, создавая элементы изображения различных оттенков и яркости. Важной характеристикой растра является его расширение. Расширение экрана-это количество точек(пикселей) на единицу длины. Чем это число выше, тем более мелкими являются сами пиксели, и, соответственно, более плотно они располагаются на плоскости, что и приводит к тому, что мы воспринимаем их как единое, цельное изображение. Из года в год разрешающая способность принтеров, мониторов, сканеров и .т.п. растёт. Для использования графических возможностей языка Паскаль необходимо в блоке описания uses подключить графический модуль Graph .Модуль

содержит набор графических функций и процедур, основные из них которые рассмотрены ниже. Положение каждой точки изображения задано координатами X и У. Координаты- целые числа, они задают номера колонки и строки и не зависят от физического размера экрана. Оси координат направлены следующим образом: горизонтальная ось X направлена слева направо; вертикальная ось У направлена сверху вниз; верхний левый угол имеет координаты(0, 0).
Рисунок 1.1
Очевидно, что запись изображения требует хранения информации о положении множества точек, для каждой из которых должен быть задан цвет. Цветное изображение получается смешиванием трех основных цветов - красного, зеленого и синего. Такая модель представления цвета называется моделью RGB (Red - Green - Blue). Управляя интенсивностью компонентов, можно получить различные оттенки и степени интенсивности цвета. В частности, для получения градаций серого надо взять интенсивности трех основных цветов равными друг другу.Стандартный модуль GraphABC системы PascalABC содержит типы, константы, переменные, процедуры и функции, позволяющие создавать изображения в специальнографическом окне.
1.3 Основные функции и процедуры работы с графикой в PascalABC

Для рисования в Pascal ABC необходимо запустить специальный модуль GraphABC, использование специальных функций и процедур помогут нарисовать точку, отрезок, окружность, прямоугольник и другие фигуры: SetPixel(x,y,color) - Закрашивает один пиксел с координатами (x,y) цветом color LineTo(x,y) - рисует отрезок от текущего положения пера до точки (x,y); координаты пера при этом также становятся равными (x,y). Line(x1,y1,x2,y2) - рисует отрезок с началом в точке (x1,y1) и концом в точке (x2,y2). SetPenColor(color) - устанавливает цвет пера, задаваемый параметром color. SetPenWidth(n) - устанавливает ширину (толщину) пера, равную n пикселям. Rectangle(x1,y1,x2,y2) - рисует прямоугольник, заданный координатами противоположных вершин (x1,y1) и (x2,y2). FloodFill(x,y,color) - заливает область одного цвета цветом color, начиная с точки (x,y). SetBrushColor(color) - устанавливает цвет кисти. Заливка кистью распространяется на замкнутый контур, описание которого следует за процедурой установки цвета кисти. Ellipse(x1,y1,x2,y2) - рисует эллипс, заданный своим описанным прямоугольником с координатами противоположных вершин (x1,y1) и (x2,y2). Circle(x,y,r) - рисует окружность с центром в точке (x,y) и радиусом r. Arc(x,y,r,a1,a2) - Рисует дугу окружности с центром в точке (x,y) и радиусом r, заключенной между двумя лучами, образующими углы a1 и a2 с осью OX (a1 и a2 - вещественные, задаются в градусах и отсчитываются против часовой стрелки).
Цвета в PascalABC:
clBlack черный clPurple фиолетовый clWhite белый clMaroon темно-красный clRed красный clNavy темно-синий clGreen зеленый clBrown коричневый clBlue синий clSkyBlue голубой clYellow желтый clCream кремовый clAqua бирюзовый clOlive оливковый clFuchsia сиреневый clTeal сине-зеленый clGray темно-серый clLime

ярко-зеленый clMoneyGreen цвет зеленых денег clLtGray светло-серый clDkGray темно-серый clMedGray серый clSilver серебряный Функции для работы с цветом:
Тип цвета Color является синонимом System.Drawing.Color.
function RGB(r,g,b: byte): Color; Возвращает цвет, который содержит красную (r), зеленую (g) и синюю (b) составляющие (r,g и b - в диапазоне от 0 до 255)
function ARGB(a,r,g,b: byte): Color; Возвращает цвет, который содержит красную (r), зеленую (g) и синюю (b) составляющие и прозрачность (a) (a,r,g,b - в диапазоне от 0 до 255)
function RedColor(r: byte): Color; Возвращает красный цвет с интенсивностью r (r - в диапазоне от 0 до 255)
function GreenColor(g: byte): Color; Возвращает зеленый цвет с интенсивностью g (g - в диапазоне от 0 до 255)
function BlueColor(b: byte): Color; Возвращает синий цвет с интенсивностью b (b - в диапазоне от 0 до 255)
function clRandom: Color; Возвращает случайный цвет
function GetRed(c: Color): integer;Возвращает красную составляющую цвета
function GetGreen(c: Color): integer; Возвращает зеленую составляющую цвета
function GetBlue(c: Color): integer; Возвращает синюю составляющую цвета.
Свойства:
property Width: integer;Ширина рисунка.
property Height: integer; Высота рисунка.
property Transparent: boolean; Прозрачность рисунка. Если Transparent=True, то при выводе рисунка его фон не отображается. Фоновым считается цвет левого нижнего пиксела рисунка.
property NeedDestroy: boolean; Определяет, должен ли рисунок разрушаться при вызове деструктора и метода Load. NeedDestroy обычно устанавливается в False, если несколько объектов классаPicture разделяют один рисунок.
Методы:
procedure Load(fname: string);
Загружает рисунок из файла с именем fname. Если NeedDestroy установлено в True, то рисунок, находившийся ранее в объекте Picture, разрушается. Рисунок с именем fname ищется вначале в текущем каталоге, а затем в подкаталоге Media\Images\ каталога программы PascalABC.exe.
procedure Save(fname: string);
Сохраняет рисунок в файле с именем fname. Формат рисунка устанавливается расширением имени файла. Допустимые расширения: .bmp, .gif, .jpg, .png.
function Handle: integer;
Возвращает описатель рисунка.
procedure Draw(x,y: integer);
Выводит рисунок в позицию (x,y) графического окна.
procedure Draw(x,y,w,h: integer);
Выводит рисунок в позицию (x,y) графического окна, масштабируя его к размеру (w,h). Если w<0 или h<0, то выводится зеркальное отражение рисунка относительно

вертикалоной или горизонтальной оси соответственно.
procedure Draw(x,y: integer; r: Rect);
Выводит часть рисунка, заключенную в прямоугольнике r, в позицию (x,y) графического окна.
procedure Draw(x,y,w,h: integer; src: Rect);
Выводит часть рисунка, заключенную в прямоугольнике r, в позицию (x,y) графического окна, масштабируя ее к размеру (w,h).
procedure CopyRect(dest: Rect; p: Picture; src: Rect);
Копирует в прямоугольник dest текущего рисунка часть рисунка p, заключенную в прямоугольнике src.
procedure FlipHorizontal;
Зеркально отображает картинку относительно горизонтальной оси симметрии.
procedure FlipVertical;Зеркально отображает картинку относительно вертикальной оси симметрии.
Рисование графических объектов
Рисование графических объектов осуществляется пером и кистью. Линии, ограничивающие объекты, рисуются пером.
Действия с пером
PenX текущая координата X пера
PenY текущая координата Y пера
SetPenColor установка цвета пера
PenColor текущий цвет пера
MoveTo перемещение пера
LineTo рисование отрезка от текущего положения пера
SetPenWidth установка ширины пера
PenWidth текущая ширина пера
SetPenStyle установка стиля пера
PenStyle текущий стиль пера
SetPenMode установка режима пера
PenMode текущий режим пера
Режим пера определяет, как цвет пера взаимодействует с цветом поверхности. В GraphABC два режима пера:
pmCopy - обычный режим: при рисовании цвет поверхности заменяется цветом пера;

pmNot - режим инвертирования: при рисовании цвет поверхности инвертируется (становится негативным), а цвет пера при этом игнорируется.
Пример:

```
uses crt,GraphABC;
var
x,x1,y,y1:longint;
begin
x:=150;
line(150,10,125,85);{строем звезду}
```

```
line(125,85,150,110);
for x1:=150 to 175 do
line(x,10,x1,110-(x1-150));
line(250,110,175,85);
for y1:=110 to 135 do
line(250,110,150+(y1-110),y1);
line(150,220,175,135);
for x1:=150 downto 125 do
line(150,220,x1,110-(x1-150));
line(40,110,125,135);
for y1:=110 downto 85 do
line(40,110,150+(y1-110),y1);
{ line(90,50,210,170);
line(210,50,90,170); дополнительное построение}
line(210,50,165,55);
line(210,50,205,95);
line(90,170,95,125);
line(90,170,135,165);
line(90,50,95,95);
line(90,50,135,55);
line(210,170,165,165);
line(210,170,205,125);
end.
```

Пример 2:

```
uses graphABC;
const
step=Pi*0.2;
Procedure DrawStar(x,y,size:integer); {x,y - координаты центра и size - радиус
снежинки}
var i,j,newsize, xnew,ynew:integer;
Begin
if size<1 then PutPixel(x,y,15) else
for i:=0 to 9 do {первый цикл - по количеству направляющих снежинки}
begin
newsize:=size;
for j:=1 to 8 do {второй цикл -рисование 8-ми подуровней снежинки}
begin
xnew:=x+round(newsize*cos(i*step));
ynew:=y+round(newsize*sin(i*step));
DrawStar(xnew,ynew,newsize div 5);
newsize:=newsize*2 div 3;
end;
end;
```

```
End;{конец процедуры}
Begin {Главная}
DrawStar(320,240,160);
End.
```

Рисунок 1.3-снежинка

2. Создание графического проекта

2.1 Понятие "фрактал"

Роль фракталов в машинной графике сегодня достаточно велика. Они приходят на помощь, например, когда требуется, с помощью нескольких коэффициентов, задать линии и поверхности очень сложной формы. С точки зрения машинной графики, фрактальная геометрия незаменима при генерации искусственных облаков, гор, поверхности моря. Фактически найден способ легкого представления сложных неевклидовых объектов, образы которых весьма похожи на природные.

Одним из основных свойств фракталов является самоподобие. В самом простом случае небольшая часть фрактала содержит информацию обо всем фрактале. Определение фрактала, данное Мандельбротом, звучит так: "Фракталом называется структура, состоящая из частей, которые в каком-то смысле подобны целому" .

1) Геометрические фракталы

Фракталы этого класса самые наглядные. В двухмерном случае их получают с помощью некоторой ломаной (или поверхности в трехмерном случае), называемой генератором. За один шаг алгоритма каждый из отрезков, составляющих ломаную, заменяется на ломаную-генератор, в соответствующем масштабе. В результате бесконечного повторения этой процедуры, получается геометрический фрактал.

Рисунок 2.1

2) Алгебраические фракталы

Это самая крупная группа фракталов. Получают их с помощью нелинейных процессов в n-мерных пространствах. Наиболее изучены двухмерные процессы. Интерпретируя нелинейный итерационный процесс, как дискретную динамическую систему, можно пользоваться терминологией теории этих систем: фазовый портрет, установившийся процесс, аттрактор и т.д.

Известно, что нелинейные динамические системы обладают несколькими устойчивыми состояниями. То состояние, в котором оказалась динамическая система после некоторого числа итераций, зависит от ее начального состояния. Поэтому каждое устойчивое состояние (или как говорят - аттрактор) обладает некоторой областью начальных состояний, из которых система обязательно попадет в рассматриваемые конечные состояния. Таким образом, фазовое пространство системы разбивается на области притяжения аттракторов. Если фазовым является двухмерное пространство, то окрашивая области притяжения различными цветами, можно получить цветовой фазовый портрет этой системы (итерационного процесса). Меняя алгоритм выбора цвета, можно получить сложные фрактальные картины с причудливыми многоцветными узорами. Неожиданностью для математиков стала

возможность с помощью примитивных алгоритмов порождать очень сложные нетривиальные структуры.
Рисунок 2.2
3) Стохастические фракталы
Еще одним известным классом фракталов являются стохастические фракталы, которые получаются в том случае, если в итерационном процессе случайным образом менять какие-либо его параметры. При этом получаются объекты очень похожие на природные - несимметричные деревья, изрезанные береговые линии и т.д. Двумерные стохастические фракталы используются при моделировании рельефа местности и поверхности моря .
Существуют и другие классификации фракталов, например деление фракталов на детерминированные (алгебраические и геометрические) и недетерминированные (стохастические).
2.2 Реализация треугольника

1) Отображается в виде треугольника, из четырех секций, каждый треугольник имеет половину ширины и высоты оригинала. Центральный треугольник инвертируется и может рассматриваться как отверстие в изображении. Каждый из трех внешних треугольников являются уменьшенной версией целого рисунка с собственными центральными отверстиями. Такая схема повторяется бесконечно.
Рисунок 2.3
2) Алгоритм построения
Существуют различные методы построения треугольника .Одним из них является использование теории хаоса с тремя аттракторами, расположенными на трех вершинах равностороннего треугольника. При переходе с текущей позиции в направлении аттрактора пройденное расстояние всегда составляет расстояние между текущей точкой и выбранной вершиной.
Рассмотрим шаги для построения фрактала:
1.Определить три точки.
2.Выбрать стартовую позицию.
3.Случайно выбрать один из трех аттракторов.
4. Переместить текущую позицию на половину в направлении позиции аттрактора и нарисовать точку.
5.Вернуться к шагу 3.
3)Рисование треугольника
Нам понадобиться три метода. Первый для генерации псевдо - случайных чисел, которые будут использоваться для определения направления во время каждой итерации. Во - вторых, нам необходимо сохранить три позиции равностороннего треугольника, для этого будем использовать структуру Point. Наконец, еще одна структура Point понадобиться для хранения текущей позиции. Объявим три переменные:
```
Random randomiser = new Random();
private Point[] points;
```

```
private Point currentLocation;
```
Размер треугольника должен быть, как можно больше, чтобы рассмотреть детали Чтобы определить наибольший возможный размер, мы должны рассматривать соотношение между высотой и шириной любого равностороннего треугольника. Если предположить, что основание треугольника выравнивается по горизонтали, ширина будет такая же, как длина одной из сторон. Высота будет иметь ширину, умноженная на половину квадратного корня из трех. Добавим метод, который рассчитает соотношение:
```
private double HeightWidthRatio()
{
return Math.Sqrt(3) / 2;
}
```
Следующий метод вычисляет размер треугольника, чтобы поместить на форме:
```
private int SideLength()
{
int height = (int)Math.Min(
(double)ClientSize.Width, ClientSize.Height / HeightWidthRatio());
// отнимаем 2 чтобы треугольник не примыкал к краям.
return (int)height - 2;
}
```
Следующий метод будет вычислять позиции каждого из трех аттракторов, каждый будет расположен в вершине треугольника. Эти значения будут основаны на размере треугольника и медиане клиентской области формы.
```
private void SetPointLocations(int sideLength)
{
Point midPoint = new Point(ClientSize.Width / 2, ClientSize.Height / 2);
points = new Point[3];
points[0] = new Point(midPoint.X
, midPoint.Y - (int)(sideLength * HeightWidthRatio() / 2));
points[1] = new Point(midPoint.X - sideLength / 2
, midPoint.Y + (int)(sideLength * HeightWidthRatio() / 2));
points[2] = new Point(midPoint.X + sideLength / 2
, midPoint.Y + (int)(sideLength * HeightWidthRatio() / 2));
}
```
Следующей задачей является создание метода (точнее двух) для размещения точек.
```
private void PlotPointLocations(Graphics g)
{
foreach (Point p in _points)
{
PlotPoint(p, g);
}
}
private void PlotPoint(Point p, Graphics g)
```

```
{
Brush b = new SolidBrush(Color.Black);
g.FillRectangle(b, p.X, p.Y, 1, 1);
}
```
Теперь напишем методы, которые используются в цикле алгоритмом для рисования фрактала. Метод DrawNextPoint будет контролировать этот процесс вызывая "MoveTowardsRandomPoint", за которым следует вызов PlotPoint, чтобы поставить точку в текущей позиции. Наконец, вызывается метод Application.DoEvents для обновления формы.
Метод MoveTowardsRandomPoint выбирает одну из трех вершин используя наш объект randomizer.Он как раз и вычисляет половину расстояния между текущей позицией и выбранным аттрактором:
```
private void DrawNextPoint(Graphics g)
{
MoveTowardsRandomPoint();
PlotPoint(currentLocation, g);
Application.DoEvents();
}
private void MoveTowardsRandomPoint()
{
int moveTowards = randomiser.Next(0,3);
currentLocation.X = (currentLocation.X + points[moveTowards].X) / 2;
currentLocation.Y = (currentLocation.Y + points[moveTowards].Y) / 2;
}
```

Метод для вызова всего что мы написали:
```
private void DrawTriangle(Graphics g)
{
int sideLength = SideLength();
SetPointLocations(sideLength);
PlotPointLocations(g);
currentLocation = new Point(points[0].X, points[0].Y);
while (working)
{
DrawNextPoint(g);
}
}
```
Поместим две кнопки для Старт и Стоп на форму и в обработчике события Click напишем:
```
private void StopButton_Click(object sender, EventArgs e)
{
StartButton.Enabled = true;
StopButton.Enabled = false;
```

```
working = false;
}
private void StartButton_Click(object sender, EventArgs e)
{
StartButton.Enabled = false;
StopButton.Enabled = true;
Graphics g = CreateGraphics();
g.Clear(Color.White);
working = true;
DrawTriangle(g);
}
```

2.3 Построения фрактала "Дерево"

Алгоритм построения "Веточки" .
Первый отрезок рисуется под произвольным углом к оси Ox, из конца данного отрезка рисуется пара отрезков, направленных в противоположные стороны от направления первого отрезка под одинаковым величиной 30 градусов. Далее рисунок повторяется с каждым из вновь построенных отрезков с уменьшением длины отрезка примерно в полтора раза.
Вложенность такого рекурсивного рисунка очевидно, не менее семи.
Рекурсивная функция имеет 4 параметра: координаты начала отрезка, его длину и угол, под некоторым рисуется отрезок к ос Ox.
Алгоритм построения состоит из следующих шагов:
1. Нарисовать отрезок.
2. Вызвать рекурсивную процедуру, рисующую левую ветку, с новыми параметрами (за координаты начала отрезка взять только что вычисленные координаты конца отрезка, за длину - длину, уменьшенную в полтора раза, за угол, уменьшенный на 30 градусов.
3. Вызвать рекурсивную процедуру, рисующую правую ветку (параметры такие же, как в пункте 2, за исключением угла - новый угол увеличивается на 30 градусов относительно первоначального угла).
4. Выход из процедуры осуществляется, когда длина ветки становится очень малой (около 2 пикселей).
В программе имеется лишь одно обращение к процедуре со следующими параметрами: координаты начала самого первого отрезка, первоначальная длина отрезка, первоначальная длина отрезка и произвольный угол наклона.
Программа "Ветка дерева" на языке PascalABC

```
uses GraphABC;
const
radian=Pi/180; deltaangle=30*radian;
Procedure Ris(x,y: integer; len, angle: real);
var
```

```
color, xnew, ynew:integer;
Begin
if len<1 then exit; {Выход из рекурсии при длине ветки меньше 1 пиксела}
if len<15 then color:=14 else color:=15;{Листья - жёлтым цветом, ствол белым}
xnew:=x+round(Len*cos(angle)); {координаты конца ветки}
ynew:=y+ round(len*sin(angle));
SetColor(color); Line(x,y,xnew,ynew); {Рисуем ветку}
Ris(xnew, ynew, len*0.65, angle-deltaangle);{Рисуем левую подветку}
Ris(xnew, ynew, len*0.65, angle+deltaangle);{Рисуем правую подветку}
end;
Begin
Ris(100,400,160,-40*radian);
End.
```

программирование графика фрактал pascal

Заключение

При написании курсовой работы рассматривался вопрос создания графических изображений в среде программирования PascalABC.

В ходе исследования мы изучили:

Основные понятия графики, функции и процедуры, возможности создания графических изображений в среде программирования PascalABC.

Полученные при исследовании знания и навыки будут полезны нам при дальнейшей работе в PascalABC, а также при работе с другими языками программирования.

Список литературы

1. Информатика(Базовый курс) С. В. Симонович, СПб: Питер, 2001г.
2. Основы программирования в задачах и примерах, А. В. Милов, Харьков: ФОЛИО, 2002г.
3. Программирование. С. Симонович, Г. Евсеев, Москва: АСТ - ПРЕСС книга 2000г.
4. Практика программирования, Ю. Кетков, А. Кетков, СПб: БХБ/ Петербург, 2002г.
5. Паскаль для школьников. С. М. Кашаев, Л. В. Шерстнева -- Москва, БХВ-Петербург, 2011 г
6. Информатика: А. Е. Споров -- Москва, АСТ, Полиграфиздат, 2010 г.
7. Основы программирования: С. Окулов -- Москва, Бином. Лаборатория знаний, 2010 г.
8. Языки программирования. Концепции и принципы: В. Ш. Кауфман -- Москва, ДМК Пресс, 2010 г.

Приложение А

Программа "Ветка дерева" на языке PascalABC

```
uses GraphABC;
const
radian=Pi/180; deltaangle=30*radian;
```

```
Procedure Ris(x,y: integer; len, angle: real);
var
color, xnew, ynew:integer;
Begin
if len<1 then exit;
{Выход из рекурсии при длине ветки меньше 1 пиксела}
if len<15 then color:=14 else color:=15;
{Листья - жёлтым цветом, ствол белым}
xnew:=x+round(Len*cos(angle));
{координаты конца ветки}
ynew:=y+ round(len*sin(angle));
SetColor(color); Line(x,y,xnew,ynew);
{Рисуем ветку}
Ris(xnew, ynew, len*0.65, angle-deltaangle);
{Рисуем левую подветку}
Ris(xnew, ynew, len*0.65, angle+deltaangle);
{Рисуем правую подветку}
end;
Begin
Ris(100,400,160,-40*radian);
End.
```

Приложение Б

Рисунок 1. Ветка дерева

Рисунок 2.Дерево