**Ткаченко Кирилл Станиславович**
*Инженер 1-й категории*
*Федерального государственного автономного образовательного учреждения высшего образования «Севастопольского государственного университета».*

# Канторова пыль

Предлагается программная реализация построения ASCII-изображения фрактала типа «Канторовы множества» («Канторова пыль») на школьном алгоритмическом языке. Программа демонстрирует работу с основными конструкциями школьного алгоритмического языка, рекурсией и многомерными массивами. Программа может быть полезна всем, увлекающимся занимательной математикой, фракталами, для изучающих программирование и школьный алгоритмический язык.

Фракталы сегодня применяются во многих областях науки и техники. Под фракталом обычно понимается самоподобный объект [1]. Части самоподобных объектов совпадают с самими объектами. Фрактал можно рассматривать как геометрическую фигуру, но состоящую из неограниченного количества частей, поскольку каждая часть этой геометрической фигуры является исходной фигурой. В свою очередь, каждая часть части геометрической фигуры также является исходной.Дальше рассмотрение происходит аналогично. На рис. 1. приведены пример фрактала [2].

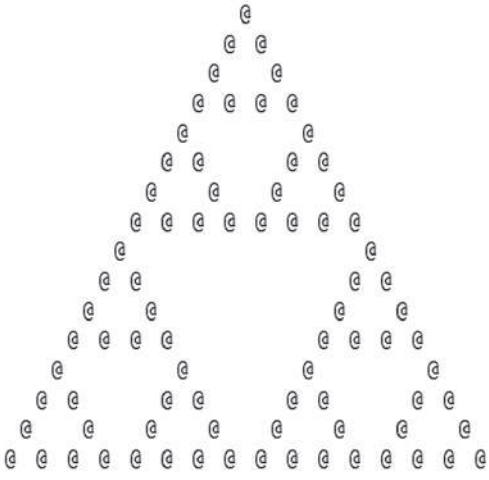

*Рис.1. Фрактал «Треугольник Серпинского»*

Одним из исторически первых рассмотренных фракталов является Канторово[1] множество или Кан-

[1] Кантор Георг ( 1845–1918) — немецкий математик, ученик Вейерштрасса. Наиболее известен как создатель теории множеств.

торова пыль [3, 4]. Пылью фрактал называется потому, что его части нельзя сосчитать, нельзя оценить его «плотность». Этот фрактал можно построить различными способами. В рамках изучения школьного алгоритмического языка для упрощения можно рассмотреть построение его ASCII-изображения по точкам. Для такого построения существует большое количество разработанных программ [5].

Классическим построением Кантора множества являются следующее рекурсивное. Из исходного «отрезка» удаляется внутренняя третья часть – внутренний сегмент. К первой и последней третям «отрезка» – сегментам – применяется классическое рекурсивное построение канторова множества. Полученные на всех рекурсивных этапах «отрезки» и являются канторовым множеством.

Для этого классического построения предлагается программа на школьном алгоритмическом языке. Вначале программы определяются ширина и высота фрактала в символах:

```
цел ШИРИНА = 81;
цел ВЫСОТА = 5;
```

Для хранения ASCII-изображения фрактала используется массив двумерный символов точки [0 : ВЫСОТА - 1, 0 : ШИРИНА - 1], первое измерение адресует отдельную строку изображения, второе – символ в рамках строки. Для удобства расчетов нумерация массива с нуля:

```
сим таб точки[0 : ВЫСОТА - 1,
0 : ШИРИНА - 1];
```

Вход в программу начинается с алгоритма Канторово Множество. Этот алгоритм производит инициализацию, формирование канторова множества, затем его вывод на стандартное устройство вывода:

```
алг КанторовоМножество
нач
  Инициализация;
  Кантор(0, ШИРИНА, 1);
  ВыводМножества;
кон
```

Алгоритм Инициализация производит инициализацию двумерного массива с изображением фрактала. Целочисленные переменные е, ё – счетчики циклов:

```
алг Инициализация
нач
  цел е, ё;
```

Каждая точка фрактала инициализируется звездочкой:

```
  нц для е от 0 до ВЫСОТА - 1
   нц для ё от 0 до ШИРИНА - 1
    точки[е, ё] := "*";
   кц
  кц
кон
```

Алгоритм Вывод Множества производит вывод двумерного массива с изображением фрактала на стандартное устройство вывода. Целочисленные переменные е, ё – счетчики циклов:

```
алг ВыводМножества
нач
  цел е, ё;
```

Фрактал построчно выводится на стандартное устройство вывода, строки разделяются терминатором:

```
  нц для е от 0 до ВЫСОТА - 1
   нц для ё от 0 до ШИРИНА - 1
    вывод точки[е, ё];
   кц
   вывод нс;
  кц
кон
```

Алгоритм Кантор (цел начало, цел длина, цел индекс) выполняет формирование отрезка канторо-

ва множества по классической рекурсивной схеме. Целочисленные аргументы используются для: начало – номер символа в строке, длина – длина отрезка, индекс – номер текущей начальной строки. Целочисленные переменные е, ё – счетчики циклов, сегмент – номер сегмента отрезка, его части:

```
алг Кантор(цел начало, цел длина, цел индекс)
нач
  цел е, ё, сегмент;
```

Определяется номер сегмента отрезка:

```
  сегмент := div(длина, 3);
```

Если текущая длина не кратна трем:

```
  если сегмент <> 0
   то
```

Для всех строк от текущей начальной до граничной:

```
     нц для е от индекс до ВЫСОТА - 1
```

По всем столбцам, принадлежащих вырезанному сегменту, точка изображения фрактала становится пробельной:

```
       нц для ё от начало + сегмент до начало + 2 * сегмент - 1
          точки[е, ё] := " ";
       кц
     кц
```

Классическая рекурсивная схема построения применяется к первой и к последней трети:

```
     Кантор(начало, сегмент, индекс + 1);
     Кантор(начало + 2 * сегмент, сегмент, индекс + 1);
  все
кон
```

```
 1  цел ШИРИНА = 81;
 2  цел ВЫСОТА = 5;
 3
 4  сим таб точки[0 : ВЫСОТА - 1, 0 : ШИРИНА - 1];
 5
 6  алг КанторовоМножество
 7  нач
 8  . Инициализация;
 9  . Кантор(0, ШИРИНА, 1);
10  . ВыводМножества;
11  кон
12
13  алг Инициализация
14  нач
15  . цел е, ё;
16  .
17  . нц для е от 0 до ВЫСОТА - 1
18  . . нц для ё от 0 до ШИРИНА - 1
19  . . . точки[е, ё] := "*";
20  . . кц
21  . кц
22  кон
23
24  алг ВыводМножества
25  нач
26  . цел е, ё;
27  .
```

```
*********************************************************************************
*****************************                 *****************************
*********         *********                   *********         *********
***   ***       ***   ***                     ***   ***       ***   ***
* *   * *       * *   * *                     * *   * *       * *   * *
```

Анализ  Выполнено шагов: 3307

*Рис. 2. Снимок экрана с результатами работы программы*

Полный исходный текст программы приводится в приложении А, результаты работы программы – в приложении Б и на рис. 2.

Разработанная на школьном алгоритмическом языке программная реализация построения ASCII-изображения фрактала типа «Канторовы множества» («Канторова пыль») демонстрирует работу с основными конструкциями школьного алгоритмического языка, рекурсией и многомерными массивами. Программа может быть полезна всем, увлекающимся занимательной математикой, фракталами, для изучающих программирование и школьный алгоритмический язык.

**Источники информации**

1. Фрактал // Википедия. URL: https://ru.wikipedia.org/wiki/Фрактал(дата обращения: 05.12.2019).
2. *Ткаченко К.С.* Реализация фрактала «Треугольник Серпинского» в 1С / К.С.Ткаченко // Системный администратор. 2019. № 4 (197). С. 60–61.
3. Канторово множество // Википедия.URL: https://ru.wikipedia.org/wiki/Канторово_множество (дата обращения: 14.11.2019).
4. Cantorset // Wikipedia.URL: https://en.wikipedia.org/wiki/Cantor_set (датаобращения: 14.11.2019).
5. Cantorset // RosettaCode.URL: https://rosettacode.org/wiki/Cantor_set (дата обращения: 14.11.2019).

## Приложение А. Полный исходный текст программы

```
цел ШИРИНА = 81;
цел ВЫСОТА = 5;

сим таб точки[0 : ВЫСОТА - 1, 0 : ШИРИНА - 1];

алг КанторовоМножество
нач
   Инициализация;
   Кантор(0, ШИРИНА, 1);
   ВыводМножества;
кон

алг Инициализация
нач
   цел е, ё;

   нц для е от 0 до ВЫСОТА - 1
     нц для ё от 0 до ШИРИНА - 1
       точки[е, ё] := "*";
     кц
   кц
кон
```

```
алг ВыводМножества
нач
   цел е, ё;

   нц для е от 0 до ВЫСОТА - 1
     нц для ё от 0 до ШИРИНА - 1
       вывод точки[е, ё];
     кц
     вывод нс;
   кц
кон

алг Кантор(цел начало, цел длина, цел индекс)
нач
   цел е, ё, сегмент;

   сегмент := div(длина, 3);

   если сегмент <> 0
     то

       нц для е от индекс до ВЫСОТА - 1
         нц для ё от начало + сегмент до начало + 2 * сегмент - 1
           точки[е, ё] := " ";
         кц
       кц

       Кантор(начало, сегмент, индекс + 1);
       Кантор(начало + 2 * сегмент, сегмент, индекс + 1);
   все
кон
```

## Приложение Б. Результаты работы программы

```
********************************************************************************
**************************                **************************
*********        *********                *********        *********
***   ***        ***   ***                ***   ***        ***   ***
* *   * *        * *   * *                * *   * *        * *   * *
```