

# Contents

1	Basic Test Results	2
2	Fractal.h	4
3	Fractal.cpp	8
4	FractalDrawer.cpp	13

# 1 Basic Test Results

```
1  Running...
2  Opening tar file
3  FractalDrawer.cpp
4  Fractal.h
5  Fractal.cpp
6  OK
7  Tar extracted O.K.
8  Checking files...
9  OK
10 Making sure files are not empty...
11 OK
12 Compilation check...
13 Compiling...
14 OK
15 Compilation seems OK! Check if you got warnings!
16
17 =====
18   Public test cases
19   =====
20
21   =====
22   Running test...
23   OK
24   Running test...
25   OK
26   Test 1 Succeeded.
27   Info: fractal number 1 with dimension x.
28   =====
29
30   =====
31   Running test...
32   OK
33   Running test...
34   OK
35   Test 2 Succeeded.
36   Info: fractal number 2 with dimension x.
37   =====
38
39   =====
40   Running test...
41   OK
42   Running test...
43   OK
44   Test 3 Succeeded.
45   Info: fractal number 3 with dimension x.
46   =====
47
48   =====
49   Running test...
50   OK
51   Running test...
52   OK
53   Test 4 Succeeded.
54   Info: few valid fractals with different dimensions.
55   =====
56
57   =====
58   Running test...
59   OK
```

```

60 Running test...
61 OK
62 Test simple_test Succeeded.
63 Info: two fractals with dimension x.
64 =====
65
66 =====
67 Running test...
68 OK
69 Running test...
70 OK
71 Test bad_cols Succeeded.
72 Info: invalid columns format of csv file.
73 =====
74
75
76 =====
77 IMPORTANT NOTICE
78 This presubmission script is NOT testing your program exit code so check it manually
79 The exit codes will be checked while grading your submission only
80 =====
81
82
83
84 =====
85 = Checking coding style =
86 =====
87 ** Total Violated Rules      : 0
88 ** Total Errors Occurs      : 0
89 ** Total Violated Files Count: 0

```

## 2 Fractal.h

```
1  //
2  // Created by brahan on 01/01/2020.
3  //
4
5  #ifndef CPP_EX2_FRACTAL_H
6  #define CPP_EX2_FRACTAL_H
7
8  #include <fstream>
9  #include <vector>
10 #include <regex>
11 #include <boost/tokenizer.hpp>
12
13
14 using std::vector;
15 //using namespace std;
16 #define INVALID "Invalid input"
17 #define TYPE_COL 0
18 #define DIM_COL 1
19
20 /**
21  * abstract class that represent general fractal object
22  */
23 class Fractal
24 {
25     #define BASE_DRAWING '#'
26     #define SPACE_DRAWING ' '
27
28 public:
29
30     /**
31      * getter for the dimension
32      * @return the fractal dimension
33      */
34     int getCurDim() const;
35
36     /**
37      * getter for the fractal template
38      * @return the fractal template
39      */
40     const vector<std::string> &getTemplate() const;
41
42     /**
43      * getter for the fractal
44      * @return the fractal
45      */
46     const vector<std::string> &getFractal() const;
47
48     /**
49      * setter for the fractal
50      * @param fractal the new fractal
51      */
52     void setFractal(const vector<std::string> &fractal);
53
54     /**
55      * draws the fractal
56      */
57     void draw();
58
59     /**
```

```

60     * if we have virtual methods we need virtual destructor
61     */
62     virtual ~Fractal() = default;
63
64     /**
65     * default copy const
66     * @param other
67     */
68     Fractal(Fractal &other) = default;
69
70 protected:
71     /**
72     * constructor for the Fractal calss
73     * @param dim the fractal dimension
74     */
75     explicit Fractal(int dim);
76
77     /**
78     * abstract function, defines the "building brick" of the current fractal
79     * @return the same type of fractal, but one dimension smaller
80     */
81     virtual Fractal *baseForm() = 0;
82
83     /**
84     * builds the fractal
85     */
86     void buildFractal();
87
88     vector<std::string> _template;
89
90 private:
91     /**
92     * helper function for the build fractal function, build fractal row
93     * @param baseForm the "building bricks" of the current fractal
94     * @param size the fractal size
95     * @param fractalLineNum which fractal line we building
96     */
97     void _rowTemplate(Fractal *baseForm, int size, int fractalLineNum);
98
99     /**
100    * helper function for the build fractal, build the spaces in the fractal
101    * @param spaceFactor the spaces
102    * @param size the fractal size
103    */
104    std::string _buildSpaceFactor(int size);
105
106    int _curDim;
107
108    vector<std::string> _fractal;
109 };
110
111 /**
112 * class that represent SierpinskiCarpet fractal object
113 */
114 class SierpinskiCarpet : public Fractal
115 {
116 public:
117     /**
118     * constructor for the class
119     * @param dim the fractal dim
120     */
121     explicit SierpinskiCarpet(int dim);
122
123 protected:
124     /**
125     * the fractal base form
126     * @return the fractal with one dimension less
127     */

```

```

128     Fractal *baseForm() override;
129
130 };
131
132 /**
133  * sub class that represent SierpinskiTriangle fractal object
134  */
135 class SierpinskiTriangle : public Fractal
136 {
137 public:
138     /**
139      * constructor for the class
140      * @param dim the fractal dim
141      */
142     explicit SierpinskiTriangle(int dim);
143
144 protected:
145     /**
146      * the fractal base form
147      * @return the fractal with one dimension less
148      */
149     Fractal *baseForm() override;
150
151
152 };
153
154 /**
155  * class that represent VicsekFractal fractal object
156  */
157 class VicsekFractal : public Fractal
158 {
159 public:
160     /**
161      * constructor for the class
162      * @param dim the fractal dim
163      */
164     explicit VicsekFractal(int dim);
165
166 protected:
167     /**
168      * the fractal base form
169      * @return the fractal with one dimension less
170      */
171     Fractal *baseForm() override;
172
173
174 };
175
176 /**
177  * factory class, will be responsible on creating the fractals according to the user input
178  */
179 class FractalFactory
180 {
181     #define CARPET 1
182     #define TRIANGLE 2
183     #define VICSEK 3
184     #define MAX_DIM 6
185     #define MIN_DIM 1
186
187 public:
188     /**
189      * static factory function
190      * @param fractalType the fractal type
191      * @param dim the fractal dimension
192      * @return a new fractal
193      */
194     static Fractal *fractalFactory(int fractalType, int dim);
195

```

```

196     /**
197     * validate the data
198     * @param data the data from the file
199     * @return 1 if the data is not valid 0 otherwise
200     */
201     static int validateData(const vector<std::vector<std::string> > &data);
202
203 };
204
205 /**
206  * will be responsible to parse the file
207  */
208 class FileHandler
209 {
210     #define LINE_PATTERN R"(\d+[, \d+[\r\n]?$)"
211     #define CSV_FILE ".csv"
212     #define FILE_SUFF_LEN 4
213     #define COMMA ","
214
215 private:
216     /**
217     * helper method that validate the line in the file
218     * @param line a single file line
219     * @param p the line regex
220     * @return 1 if there is error, 0 else
221     */
222     static int _validateLine(const std::string &line, const std::regex &p);
223
224 public:
225     /**
226     * parse the file data
227     * @param path the file path
228     * @param isValid indicator if the file is valid
229     * @return vector with the file data if everything ok
230     */
231     static std::vector<std::vector<std::string> > parseFile(const std::string &path, int &isValid);
232 };
233
234 #endif //CPP_EX2_FRACTAL_H

```

## 3 Fractal.cpp

```
1  //
2  // Created by brahan on 01/01/2020.
3  //
4  #include <iostream>
5  #include <cmath>
6  #include <boost/tokenizer.hpp>
7  #include "Fractal.h"
8
9  /**
10   * abstract function, defines the "building brick" of the current fractal
11   * @return the same type of fractal, but one dimension smaller
12   */
13  Fractal::Fractal(int dim) :
14      _curDim(dim)
15  {
16
17  }
18
19  /**
20   * getter for the fractal
21   * @return the fractal dim
22   */
23  int Fractal::getCurDim() const
24  {
25      return _curDim;
26  }
27
28  /**
29   * getter for the fractal
30   * @return the template
31   */
32  const vector<std::string> &Fractal::getTemplate() const
33  {
34      return _template;
35  }
36
37  /**
38   * getter for the fractal
39   * @return the fractal
40   */
41  const vector<std::string> &Fractal::getFractal() const
42  {
43      return _fractal;
44  }
45
46  /**
47   * draws the fractal
48   */
49  void Fractal::draw()
50  {
51      for (const auto &cell : this->_fractal)
52      {
53          std::cout << cell;
54          std::cout << std::endl;
55      }
56  }
57
58  /**
59   * builds the fractal
```



```

60  */
61  void Fractal::buildFractal()
62  {
63      if (getCurDim() == MIN_DIM)
64      {
65          setFractal(getTemplate());
66          return;
67      }
68      Fractal *base = baseForm();
69      size_t size = pow(getTemplate().size(), getCurDim() - 1);
70      for (int j = 0; j < (int) getTemplate().size(); ++j)
71      {
72          _rowTemplate(base, size, j);
73      }
74      delete (base);
75  }
76
77  /**
78   * setter for the fractal
79   * @param fractal the new fractal
80   */
81  void Fractal::setFractal(const vector<std::string> &fractal)
82  {
83      _fractal = fractal;
84  }
85
86  /**
87   * helper function for the build fractal function, build fractal row
88   * @param baseForm the "building bricks" of the current fractal
89   * @param size the fractal size
90   * @param fractalLineNum which fractal line we building
91   */
92  void Fractal::_rowTemplate(Fractal *baseForm, int size, int fractalLineNum)
93  {
94      std::string fractalRow;
95      std::string spaceFactor = _buildSpaceFactor(size);
96      for (int i = 0; i < size; ++i)
97      {
98          for (int j = 0; j < (int) getTemplate().size(); ++j)
99          {
100              if (getTemplate()[fractalLineNum][j] == BASE_DRAWING) //if we have # we need to put baseform instead
101              {
102                  fractalRow += baseForm->getFractal().at(i);
103              }
104              else // else we have space, so we need to put the spaces according the the fractal dim
105              {
106                  fractalRow += spaceFactor;
107              }
108          }
109          this->_fractal.push_back(fractalRow);
110          fractalRow = "";
111      }
112  }
113
114  std::string Fractal::_buildSpaceFactor(int size)
115  {
116      std::string spaceFactor;
117      for (int i = 0; i < (int) size; ++i)
118      {
119          spaceFactor += SPACE_DRAWING;
120      }
121      return spaceFactor;
122  }
123
124  /**
125   * constructor for the class
126   * @param dim the fractal dim
127   */

```

```

128 SierpinskiCarpet::SierpinskiCarpet(int dim) : Fractal(dim)
129 {
130     this->_template = {"###", "# #", "###"};
131     (dim == MIN_DIM) ? setFractal(getTemplate()) : buildFractal();
132 }
133
134 /**
135  * constructor for the class
136  * @param dim the fractal dim
137  */
138 SierpinskiTriangle::SierpinskiTriangle(int dim) : Fractal(dim)
139 {
140     this->_template = {"##", "# "};
141     (dim == MIN_DIM) ? setFractal(getTemplate()) : buildFractal();
142 }
143
144 /**
145  * constructor for the class
146  * @param dim the fractal dim
147  */
148 VicsekFractal::VicsekFractal(int dim) : Fractal(dim)
149 {
150     this->_template = {"# #", "# #", "# #"};
151     (dim == MIN_DIM) ? setFractal(getTemplate()) : buildFractal();
152 }
153
154 /**
155  * the fractal base form
156  * @return the fractal with one dimension less
157  */
158 Fractal *SierpinskiCarpet::baseForm()
159 {
160     return new SierpinskiCarpet(getCurDim() - 1);
161 }
162
163 /**
164  * the fractal base form
165  * @return the fractal with one dimension less
166  */
167 Fractal *SierpinskiTriangle::baseForm()
168 {
169     return new SierpinskiTriangle(getCurDim() - 1);
170 }
171
172 /**
173  * the fractal base form
174  * @return the fractal with one dimension less
175  */
176 Fractal *VicsekFractal::baseForm()
177 {
178     return new VicsekFractal(getCurDim() - 1);
179 }
180
181 /**
182  * static factory function
183  * @param fractalType the fractal type
184  * @param dim the fractal dimension
185  * @return a new fractal
186  */
187 Fractal *FractalFactory::fractalFactory(int fractalType, int dim)
188 {
189     switch (fractalType)
190     {
191         case CARPET:
192             return new SierpinskiCarpet(dim);
193         case TRIANGLE:
194             return new SierpinskiTriangle(dim);
195         case VICSEK:

```

```

196         return new VicsekFractal(dim);
197     default:
198         return nullptr;
199     }
200 }
201
202 /**
203  * validate the data
204  * @param data the data from the file
205  * @return 1 if the data is not valid 0 otherwise
206  */
207 int FractalFactory::validateData(const std::vector<std::vector<std::string> > &data)
208 {
209     for (auto &i : data)
210     {
211         int type = stoi(i.at(TYPE_COL));
212         int dim = stoi(i.at(DIM_COL));
213         if (type < CARPET || type > VICSEK || dim < MIN_DIM || dim > MAX_DIM)
214         {
215             return EXIT_FAILURE;
216         }
217     }
218     return EXIT_SUCCESS;
219 }
220
221 /**
222  * helper method that validate the line in the file
223  * @param line a single file line
224  * @param p the line regex
225  * @return 1 if there is error, 0 else
226  */
227 int FileHandler::_validateLine(const std::string &line, const std::regex &p)
228 {
229     if (line.empty())
230     {
231         return EXIT_FAILURE;
232     }
233     if (!regex_match(line, p))
234     {
235         return EXIT_FAILURE;
236     }
237     return EXIT_SUCCESS;
238 }
239
240 /**
241  * parse the file data
242  * @param path the file path
243  * @param isValid indicator if the file is valid
244  * @return vector with the file data if everything ok
245  */
246 std::vector<std::vector<std::string> > FileHandler::parseFile(const std::string &path, int &isValid)
247 {
248     std::ifstream file(path);
249     std::regex p(LINE_PATTERN);
250     if (path.compare(path.size() - FILE_SUFF_LEN, FILE_SUFF_LEN, CSV_FILE))
251     {
252         isValid = EXIT_FAILURE;
253     }
254     if (file.bad() || file.fail())
255     {
256         file.close();
257         isValid = EXIT_FAILURE;
258     }
259     boost::char_separator<char> separator{COMMA};
260     std::vector<std::vector<std::string> > dataList;
261     std::string line;
262     while (getline(file, line))
263     {

```

```

264         if (_validateLine(line, p))
265         {
266             file.close();
267             isValid = EXIT_FAILURE;
268             break;
269         }
270         boost::tokenizer<boost::char_separator<char>> pattern{line, separator};
271         std::vector<std::string> parsedLine;
272         for (const auto &t:pattern)
273         {
274             parsedLine.push_back(t);
275         }
276         dataList.push_back(parsedLine);
277     }
278     file.close();
279     return dataList;
280 }

```

## 4 FractalDrawer.cpp

```
1  /**
2   * @file FractalDrawer.c
3   * @author Brahan Wassan <brahan>
4   * @version 1.0
5   * @date 8 Jan 2020
6   *
7   * @brief Program that build a tree from a given txt file
8   *
9   * @section DESCRIPTION
10  * The program draws fractals from csv file
11  * Input : csv file with integer that represent the fractal type and size
12  * program build explanation: i created abstract class - Fractal that defines the basic fractal features
13  * there are 3 type of fractals, all of them are subclass of Fractal, and they override the method buildFractal
14  * in addition to those 4 classes, to match the single resoponsibilty prinsible i created a factory class which
15  * is responsible on creating the fractals, and file handler class which handles the file validation and parsing.
16  */
17 #include <iostream>
18 #include "Fractal.h"
19
20 #define USAGE "Usage: FractalDrawer <file path>"
21 #define NUM_ARGS 2
22 #define FILE_IDX 1
23
24 /**
25  * the function draws all the fractals
26  * @param data the data from the file
27  */
28 void drawAll(std::vector<std::vector<std::string> > data)
29 {
30     for (int j = (int) data.size() - 1; j >= 0; --j)
31     {
32         Fractal *fractal = FractalFactory::fractalFactory(stoi(data.at(j).at(TYPE_COL)), stoi(data.at(j).at(DIM_COL)));
33         fractal->draw();
34         std::cout << std::endl;
35         delete (fractal);
36     }
37 }
38
39 /**
40  * main function
41  * @param argc num of arg
42  * @param argv program args
43  * @return 1 if error, 0 if not
44  */
45 int main(int argc, const char *argv[])
46 {
47     if (argc != NUM_ARGS)
48     {
49         std::cerr << USAGE << std::endl;
50         return EXIT_FAILURE;
51     }
52     int isValid = 0;
53     std::vector<std::vector<std::string> > data = FileHandler::parseFile(argv[FILE_IDX], isValid);
54     if (isValid)
55     {
56         std::cerr << INVALID << std::endl;
57         return EXIT_FAILURE;
58     }
59     isValid = FractalFactory::validateData(data);
```

```
60     if (isValid)
61     {
62         std::cerr << INVALID << std::endl;
63         return EXIT_FAILURE;
64     }
65     drawAll(data);
66     return EXIT_SUCCESS;
67 }
```