Contents

1 Basic Test Results 2

2 Fractal.h 4

3 Fractal.cpp 8

4 FractalDrawer.cpp 13

1 Basic Test Results

```
Running...
Opening tar file
FractalDrawer.cpp
Fractal.h
Fractal.cpp
OK
Tar extracted O.K.
Checking files...
OK
Making sure files are not empty...
OK
Compilation check...
Compiling...
OK
Compilation seems OK! Check if you got warnings!

=====================
 Public test cases
=====================

=====================
Running test...
OK
Running test...
OK
Test 1 Succeeded.
Info: fractal number 1 with dimension x.
=====================

=====================
Running test...
OK
Running test...
OK
Test 2 Succeeded.
Info: fractal number 2 with dimension x.
=====================

=====================
Running test...
OK
Running test...
OK
Test 3 Succeeded.
Info: fractal number 3 with dimension x.
=====================

=====================
Running test...
OK
Running test...
OK
Test 4 Succeeded.
Info: few valid fractals with different dimensions.
=====================

=====================
Running test...
OK
```

```
Running test...
OK
Test simple_test Succeeded.
Info: two fractals with dimension x.
=====================

=====================
Running test...
OK
Running test...
OK
Test bad_cols Succeeded.
Info: invalid columns format of csv file.
=====================


=====================
 IMPORTANT NOTICE
  This presubmission script is NOT testing your program exit code so check it manually
  The exit codes will be checked while grading your submission only
=====================



========================
= Checking coding style =
========================
 ** Total Violated Rules      : 0
 ** Total Errors Occurs       : 0
 ** Total Violated Files Count: 0
```

2 Fractal.h

```cpp
//
// Created by brahan on 01/01/2020.
//

#ifndef CPP_EX2_FRACTAL_H
#define CPP_EX2_FRACTAL_H

#include <fstream>
#include <vector>
#include <regex>
#include <boost/tokenizer.hpp>


using std::vector;
//using namespace std;
#define INVALID "Invalid input"
#define TYPE_COL 0
#define DIM_COL 1

/**
 * abstract class that represent general fractal object
 */
class Fractal
{
#define BASE_DRAWING '#'
#define SPACE_DRAWING ' '

public:

    /**
     * getter for the dimension
     * @return the fractal dimension
     */
    int getCurDim() const;

    /**
     * getter for the fractal template
     * @return the fractal template
     */
    const vector<std::string> &getTemplate() const;

    /**
     * getter for the fractal
     * @return the fractal
     */
    const vector<std::string> &getFractal() const;

    /**
     * setter for the fractal
     * @param fractal the new fractal
     */
    void setFractal(const vector<std::string> &fractal);

    /**
     * draws the fractal
     */
    void draw();

    /**
```

```
     * if we have virtual methods we need virtual destructor
     */
    virtual ~Fractal() = default;

    /**
     * default copy const
     * @param other
     */
    Fractal(Fractal &other) = default;

protected:
    /**
     * constructor for the Fractal calss
     * @param dim the fractal dimension
     */
    explicit Fractal(int dim);

    /**
     * abstract function, defines the "building brick" of the current fractal
     * @return the same type of fractal, but one dimension smaller
     */
    virtual Fractal *baseForm() = 0;

    /**
     * builds the fractal
     */
    void buildFractal();

    vector<std::string> _template;

private:
    /**
     * helper function for the build fractal function, build fractal row
     * @param baseForm the "building bricks" of the current fractal
     * @param size the fractal size
     * @param fractalLineNum which fractal line we building
     */
    void _rowTemplate(Fractal *baseForm, int size, int fractalLineNum);

    /**
     * helper function for the build fractal, build the spaces in the fractal
     * @param spaceFactor the spaces
     * @param size the fractal size
     */
    std::string _buildSpaceFactor(int size);

    int _curDim;

    vector<std::string> _fractal;
};

/**
 * class that represent SierpinskiCarpet fractal object
 */
class SierpinskiCarpet : public Fractal
{
public:
    /**
     * constructor for the class
     * @param dim the fractal dim
     */
    explicit SierpinskiCarpet(int dim);

protected:
    /**
     * the fractal base form
     * @return the fractal with one dimension less
     */
```

```
    Fractal *baseForm() override;

};

/**
 * sub class that represent SierpinskiTriangle fractal object
 */
class SierpinskiTriangle : public Fractal
{
public:
    /**
     * constructor for the class
     * @param dim the fractal dim
     */
    explicit SierpinskiTriangle(int dim);

protected:
    /**
     * the fractal base form
     * @return the fractal with one dimension less
     */
    Fractal *baseForm() override;


};

/**
 * class that represent VicsekFractal fractal object
 */
class VicsekFractal : public Fractal
{
public:
    /**
     * constructor for the class
     * @param dim the fractal dim
     */
    explicit VicsekFractal(int dim);

protected:
    /**
     * the fractal base form
     * @return the fractal with one dimension less
     */
    Fractal *baseForm() override;


};

/**
 * factory class, will be responsible on creating the fractals according to the user input
 */
class FractalFactory
{
#define CARPET 1
#define TRIANGLE 2
#define VICSEK 3
#define MAX_DIM 6
#define MIN_DIM 1

public:
    /**
     * static factory function
     * @param fractalType the fractal type
     * @param dim the fractal dimension
     * @return a new fractal
     */
    static Fractal *fractalFactory(int fractalType, int dim);

```

```
    /**
     * validate the data
     * @param data the data from the file
     * @return 1 if the data is not valid 0 otherwise
     */
    static int validateData(const vector<std::vector<std::string> > &data);

};

/**
 * will be responsible to parse the file
 */
class FileHandler
{
#define LINE_PATTERN R"(^\d+\,\d+[\r\n]?$)"
#define CSV_FILE ".csv"
#define FILE_SUFF_LEN 4
#define COMMA ","

private:
    /**
     * helper method that validate the line in the file
     * @param line a single file line
     * @param p the line regex
     * @return 1 if there is error, 0 else
     */
    static int _validateLine(const std::string &line, const std::regex &p);

public:
    /**
     * parse the file data
     * @param path the file path
     * @param isValid indicator if the file is valid
     * @return vector with the file data if everything ok
     */
    static std::vector<std::vector<std::string> > parseFile(const std::string &path, int &isValid);
};

#endif //CPP_EX2_FRACTAL_H
```

3 Fractal.cpp

```cpp
//
// Created by brahan on 01/01/2020.
//
#include <iostream>
#include <cmath>
#include <boost/tokenizer.hpp>
#include "Fractal.h"

/**
 * abstract function, defines the "building brick" of the current fractal
 * @return the same type of fractal, but one dimension smaller
 */
Fractal::Fractal(int dim) :
        _curDim(dim)
{

}

/**
 * getter for the fractal
 * @return the fractal dim
 */
int Fractal::getCurDim() const
{
    return _curDim;
}

/**
 * getter for the fractal
 * @return the template
 */
const vector<std::string> &Fractal::getTemplate() const
{
    return _template;
}

/**
 * getter for the fractal
 * @return the fractal
 */
const vector<std::string> &Fractal::getFractal() const
{
    return _fractal;
}

/**
 * draws the fractal
 */
void Fractal::draw()
{
    for (const auto &cell : this->_fractal)
    {
        std::cout << cell;
        std::cout << std::endl;
    }
}

/**
 * builds the fractal
```

```
 */
void Fractal::buildFractal()
{
    if (getCurDim() == MIN_DIM)
    {
        setFractal(getTemplate());
        return;
    }
    Fractal *base = baseForm();
    size_t size = pow(getTemplate().size(), getCurDim() - 1);
    for (int j = 0; j < (int) getTemplate().size(); ++j)
    {
        _rowTemplate(base, size, j);
    }
    delete (base);
}

/**
 * setter for the fractal
 * @param fractal the new fractal
 */
void Fractal::setFractal(const vector<std::string> &fractal)
{
    _fractal = fractal;
}

/**
 * helper function for the build fractal function, build fractal row
 * @param baseForm the "building bricks" of the current fractal
 * @param size the fractal size
 * @param fractalLineNum which fractal line we building
 */
void Fractal::_rowTemplate(Fractal *baseForm, int size, int fractalLineNum)
{
    std::string fractalRow;
    std::string spaceFactor = _buildSpaceFactor(size);
    for (int i = 0; i < size; ++i)
    {
        for (int j = 0; j < (int) getTemplate().size(); ++j)
        {
            if (getTemplate()[fractalLineNum][j] == BASE_DRAWING)//if we have # we need to put baseform instead
            {
                fractalRow += baseForm->getFractal().at(i);
            }
            else // else we have space, so we need to put the spaces according the the fractal dim
            {
                fractalRow += spaceFactor;
            }
        }
        this->_fractal.push_back(fractalRow);
        fractalRow = "";
    }
}

std::string Fractal::_buildSpaceFactor(int size)
{
    std::string spaceFactor;
    for (int i = 0; i < (int) size; ++i)
    {
        spaceFactor += SPACE_DRAWING;
    }
    return spaceFactor;
}

/**
 * constructor for the class
 * @param dim the fractal dim
 */
```

```
SierpinskiCarpet::SierpinskiCarpet(int dim) : Fractal(dim)
{
    this->_template = {"###", "# #", "###"};
    (dim == MIN_DIM) ? setFractal(getTemplate()) : buildFractal();
}

/**
 * constructor for the class
 * @param dim the fractal dim
 */
SierpinskiTriangle::SierpinskiTriangle(int dim) : Fractal(dim)
{
    this->_template = {"##", "# "};
    (dim == MIN_DIM) ? setFractal(getTemplate()) : buildFractal();
}

/**
 * constructor for the class
 * @param dim the fractal dim
 */
VicsekFractal::VicsekFractal(int dim) : Fractal(dim)
{
    this->_template = {"# #", " # ", "# #"};
    (dim == MIN_DIM) ? setFractal(getTemplate()) : buildFractal();
}

/**
 * the fractal base form
 * @return the fractal with one dimension less
 */
Fractal *SierpinskiCarpet::baseForm()
{
    return new SierpinskiCarpet(getCurDim() - 1);
}

/**
 * the fractal base form
 * @return the fractal with one dimension less
 */
Fractal *SierpinskiTriangle::baseForm()
{
    return new SierpinskiTriangle(getCurDim() - 1);
}

/**
 * the fractal base form
 * @return the fractal with one dimension less
 */
Fractal *VicsekFractal::baseForm()
{
    return new VicsekFractal(getCurDim() - 1);
}

/**
 * static factory function
 * @param fractalType the fractal type
 * @param dim the fractal dimension
 * @return a new fractal
 */
Fractal *FractalFactory::fractalFactory(int fractalType, int dim)
{
    switch (fractalType)
    {
        case CARPET:
            return new SierpinskiCarpet(dim);
        case TRIANGLE:
            return new SierpinskiTriangle(dim);
        case VICSEK:
```

```
            return new VicsekFractal(dim);
        default:
            return nullptr;
    }
}

/**
 * validate the data
 * @param data the data from the file
 * @return 1 if the data is not valid 0 otherwise
 */
int FractalFactory::validateData(const std::vector<std::vector<std::string> > &data)
{
    for (auto &i : data)
    {
        int type = stoi(i.at(TYPE_COL));
        int dim = stoi(i.at(DIM_COL));
        if (type < CARPET || type > VICSEK || dim < MIN_DIM || dim > MAX_DIM)
        {
            return EXIT_FAILURE;
        }
    }
    return EXIT_SUCCESS;
}

/**
 * helper method that validate the line in the file
 * @param line a single file line
 * @param p the line regex
 * @return 1 if there is error, 0 else
 */
int FileHandler::_validateLine(const std::string &line, const std::regex &p)
{
    if (line.empty())
    {
        return EXIT_FAILURE;
    }
    if (!regex_match(line, p))
    {
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}

/**
 * parse the file data
 * @param path the file path
 * @param isValid indicator if the file is valid
 * @return vector with the file data if everything ok
 */
std::vector<std::vector<std::string> > FileHandler::parseFile(const std::string &path, int &isValid)
{
    std::ifstream file(path);
    std::regex p(LINE_PATTERN);
    if (path.compare(path.size() - FILE_SUFF_LEN, FILE_SUFF_LEN, CSV_FILE))
    {
        isValid = EXIT_FAILURE;
    }
    if (file.bad() || file.fail())
    {
        file.close();
        isValid = EXIT_FAILURE;
    }
    boost::char_separator<char> separator{COMMA};
    std::vector<std::vector<std::string> > dataList;
    std::string line;
    while (getline(file, line))
    {
```

```
        if (_validateLine(line, p))
        {
            file.close();
            isValid = EXIT_FAILURE;
            break;
        }
        boost::tokenizer<boost::char_separator<char>> pattern{line, separator};
        std::vector<std::string> parsedLine;
        for (const auto &t:pattern)
        {
            parsedLine.push_back(t);
        }
        dataList.push_back(parsedLine);
    }
    file.close();
    return dataList;
}
```

4 FractalDrawer.cpp

```cpp
/**
 * @file FractalDrawer.c
 * @author  Brahan Wassan <brahan>
 * @version 1.0
 * @date 8 Jan 2020
 *
 * @brief Program that build a tree from a given txt file
 *
 * @section DESCRIPTION
 * The program draws fractals from csv file
 * Input  : csv file with integer that represent the fractal type and size
 * program build explanation: i created abstract class - Fractal that  defines the basic fractal features
 * there are 3 type of fractals, all of them are subclass of Fractal, and they override the method buildFractal
 * in addition to those 4 classes, to match the single resoponsibilty prinsible i created a factory class which
 * is responsible on creating the fractals, and file handler class which handles the file validation and parsing.
 */
#include <iostream>
#include "Fractal.h"

#define USAGE "Usage: FractalDrawer <file path>"
#define NUM_ARGS 2
#define FILE_IDX 1

/**
 * the function draws all the fractals
 * @param data the data from the file
 */
void drawAll(std::vector<std::vector<std::string> > data)
{
    for (int j = (int) data.size() - 1; j >= 0; --j)
    {
        Fractal *fractal = FractalFactory::fractalFactory(stoi(data.at(j).at(TYPE_COL)), stoi(data.at(j).at(DIM_COL)));
        fractal->draw();
        std::cout << std::endl;
        delete (fractal);
    }
}

/**
 * main function
 * @param argc num of arg
 * @param argv program args
 * @return 1 if error, 0 if not
 */
int main(int argc, const char *argv[])
{
    if (argc != NUM_ARGS)
    {
        std::cerr << USAGE << std::endl;
        return EXIT_FAILURE;
    }
    int isValid = 0;
    std::vector<std::vector<std::string> > data = FileHandler::parseFile(argv[FILE_IDX], isValid);
    if (isValid)
    {
        std::cerr << INVALID << std::endl;
        return EXIT_FAILURE;
    }
    isValid = FractalFactory::validateData(data);
```

```
    if (isValid)
    {
        std::cerr << INVALID << std::endl;
        return EXIT_FAILURE;
    }
    drawAll(data);
    return EXIT_SUCCESS;
}
```