ФИО: Ребдев Павел Александрович
Группа: 5130904/30008
Лабораторная работа: «Массивы»

Постановка задачи

Разработать детальные требования и тест план для следующей задачи:

[CNT-LOC-MAX]. Реализовать программу рассчитывающую количество локальных максимумов в матрице записанной в файле

Детальные требования

1. Параметры командной строки должны быть заданы корректно:
   1.1. Если количество параметров командной строки не равно 3, то программа завершается с кодом возврата 1
   1.2. Если первый параметр командной строки не равен 1 или 2, то программа завершается с кодом возврата 1 и соответствующим сообщением об ошибке в стандартном потоке ошибок
2. Содержимое файла должно соответствовать матрице:
   2.1. Если файл пуст, то программа завершается с кодом возврата 2 и соответствующим сообщением об ошибке в стандартном потоке ошибок
   2.2. Если количество элементов в матрице меньше произведения количества строк на количество столбцов, то программа завершается с кодом возврата 2
   2.3. Все элементы из файла, включая количество строк и столбцов являются числами. Если хотя бы один из элементов в файле не является числом, то программа завершается с кодом возврата 2 и соответствующим сообщением об ошибке в стандартном потоке ошибок
3. Размер матрицы и элементы заданы корректно:
   3.1. Если размер матрицы и элементы заданы корректно, то программа должна завершится с выводом в консоль сообщения, содержащего количество локальных максимумов и кодом возврата 0

Тест-план

Проверка детальных требований с помощью тест-плана:

#	Описание	Результат
1.1	Если количество параметров командной строки не равно 3, то программа завершается с кодом возврата 1	**ARGV:** `1 abc input.txt output.txt` **Expected:** **exit code:** 1

1.2	Если первый параметр командной строки не равен 1 или 2, то программа завершается с кодом возврата 1 и соответствующим сообщением об ошибке в стандартном потоке ошибок	**ARGV:**
		`3 input.txt output.txt`
		Expected:
		`First param must be only number`
		exit code: 1

2.1	Если файл пуст, то программа завершается с кодом возврата 2 и соответствующим сообщением об ошибке в стандартном потоке ошибок	**Input file:** **Expected:** `File is empty` **exit code:** 2
2.2	Если количество элементов в матрице меньше произведения количества строк на количество столбцов, то программа завершается с кодом возврата 2	**Input file:** `2 3 1 2 3 4 5` **Expected:** **exit code**: 2
2.3	Все элементы из файла, включая количество строк и столбцов являются числами. Если хотя бы один из элементов в файле не является числом, то программа завершается с кодом возврата 2 и соответствующим сообщением об ошибке в стандартном потоке ошибок	**Input file:** `2 3 1 2 3 f 4 5` **Expected:** **exit code**: 2
3.1	Если размер матрицы и элементы заданы	**Input file:** `3 3 1 7 1 5 1 8 1 9 1`

	корректно, то программа должна завершится с выводом в консоль сообщения, содержащего количество локальных максимумов и кодом возврата 0	**Expected:** 0 **exit code:** 0
		Input file: 5 4 1 1 1 1 1 1 9 1 1 8 1 1 1 1 9 1 1 1 1 1 **Expected:** 2 **exit code:** 0

Исходные тексты программы

Файлы с исходными текстами лабораторной работы (полагаем <ROOT> для папки в котором располагаются исходные тексты):

./<ROOT>/main.cpp

```
#include <limits>
#include <iostream>
#include <cstring>

#include "matrix.hpp"
#include "readFromFile.hpp"

int main(int argc, char ** argv)
{
  if (argc != 4)
  {
    return 1;
  }

  int arrayMode = 0;
  try
  {
    arrayMode = std::stoi(argv[1]);
  }
  catch (const std::logic_error & e)
  {
    std::cerr << "First param isn't number!\n";
    return 1;
  }

  if (((arrayMode != 1) && (arrayMode != 2)) || (argv[1][1] != 0))
  {
    std::cerr << "First param must be only number!\n";
    return 1;
  }

  std::ifstream inputFile(argv[2]);
  if (!inputFile.is_open())
  {
    return 2;
  }

  size_t rows = 0, colums = 0;
  try
  {
```

```
    inputFile >> rows;

    if (inputFile.eof())
    {
      std::cerr << "File is empty!\n";
      return 2;
    }

    inputFile >> colums;

    if (!inputFile)
    {
      return 2;
    }
  }
  catch (const std::logic_error & e)
  {
    return 2;
  }
  if ((rows == 0) || (colums == 0))
  {
    return (((rows || colums) == 0) ? 0 : 2);
  }

  if (rows > (std::numeric_limits< size_t >::max() / colums))
  {
    return 2;
  }

  long long int arr2[10000] = {0};
  long long int * array = nullptr;

  if (arrayMode == 1)
  {
    array = arr2;
  }
  else
  {
    array = new long long int[rows * colums];
  }

  try
  {
        size_t  fillIsOk  =  rebdev::fillTheMatrix(array,  rows,  colums,
inputFile);
    if (fillIsOk < (colums * rows))
    {
      std::cerr << fillIsOk << " elements is writing\n";
      if (arrayMode != 1)
      {
        delete [] array;
      }
      return 2;
    }
```

```
    std::ofstream outputFile(argv[3]);

    if (!outputFile.is_open())
    {
      if (arrayMode != 1)
      {
        delete [] array;
      }
      return 2;
    }

    outputFile << rebdev::findNumberOfLocalMax(array, rows, colums);
    if (arrayMode != 1)
    {
      delete [] array;
    }
  }
  catch (const std::logic_error & e)
  {
    delete [] array;
    return 2;
  }

  return 0;
}
```

./<ROOT>/matrix.hpp

```
#ifndef MATRIX_HPP
#define MATRIX_HPP

#include <fstream>
#include <cstddef>

namespace rebdev
{
   size_t fillTheMatrix(long long int * array, size_t rows, size_t
colums, std::ifstream & inputFile);
  size_t findNumberOfLocalMax(long long int * arr, size_t rows, size_t
colums);
}
#endif
```

./<ROOT>/matrix.cpp

```
#include <limits>
#include <fstream>
#include "matrix.hpp"
#include "readFromFile.hpp"

size_t rebdev::fillTheMatrix(long long int * array, size_t rows, size_t
colums, std::ifstream & inputFile)
{
  for (size_t i = 0; i < (colums * rows); ++i)
  {
```

```
    if (!(inputFile >> array[i]))
    {
      return i;
    }
  }
  return (colums * rows);
}

size_t rebdev::findNumberOfLocalMax(long long int * arr, size_t rows,
size_t colums)
{
  size_t numberOfLocalMax = 0;
  bool isLocalMax = 1;
  for (size_t i = 0; i < (colums * (rows-1)); ++i)
  {
     if (((i % colums) != 0) && ((i % colums) != (colums - 1)) && (i >
(colums-1)))
    {
      isLocalMax = 1;
      for (int rowIndex = -1; rowIndex <= 1; ++rowIndex)
      {
        for (int columIndex = -1; columIndex <= 1; ++columIndex)
        {
          if ((rowIndex != 0) && (columIndex != 0))
          {
            if (arr[i] <= arr[i + colums * rowIndex + columIndex])
            {
              isLocalMax = 0;
              columIndex = 2;
              rowIndex = 2;
              break;
            }
          }
        }
      }
      numberOfLocalMax += isLocalMax;
    }
  }

  return numberOfLocalMax;
}
```

./<ROOT>/readFromFile.hpp

```
#ifndef READFROMFILE_HPP
#define READFROMFILE_HPP
#include <fstream>
namespace rebdev
{
  long long int readFromFileLLI(std::ifstream & inputFile);
}
#endif
```

./<ROOT>/readFromFile.cpp

```
#include "readFromFile.hpp"
#include <stdexcept>
```

```
#include <fstream>

long long int rebdev::readFromFileLLI(std::ifstream & inputFile)
{
  long long int num = 0;
  inputFile >> num;

  if (!inputFile)
  {
    throw std::logic_error("Reading error: can't read from file");
  }
  else if (inputFile.eof())
  {
    throw std::logic_error("Reading error: file is end");
  }

  return num;
}
```