Mittuniversitetet
MID SWEDEN UNIVERSITY

Lab 4 – Exceptions, Templates, and streams

Object-oriented programming in C ++

Objective: Implementation of exception handling in combination with templates and IO streams.

Lab 4 – Exceptions, Templates, and streams

Background:
Assume the following scenario: Measurements from an external sensor are stored as numerical values in a file on disk. The measurements and the transmission is very sensitive which leads to the stored measured values containing two types of errors:

1. Sensor error that causes the value outside a specified range.
2. Transmission error that causes the sings other than numbers written to the file.

Assignment:
Your task is to write a program that filters the measurements in two stages. First, filter the numbers that cannot be read because they contain invalid characters, and then filter out the numbers that are outside the accepted interval.

The values to be processed are presented in the file `values.dat`. The numbers are written in text format and are separated by newline-sign. The values will be is converted to type double in the program. In the filtering process exception, handling will be used.

The reading of values from `values.dat` and filtering of incorrect signs will be done with an instance of the class `DataFileReader`:

```
template<typename T>
class DataFileReader {

public:
  DataFileReader(string aDataFileName, string aErrorFileName);
  /* pre: A file named aDataFile contains values to read.
  */

  ~DataFileReader();
  /* post: Files are closed */

  void openFiles();
  /* post:  An input stream from the file named aDataFile and
      an output stream to the file named aErrorFile are
      opened. If either of these operations fails a
      runtime_error exception is thrown. */

  bool readNextValue(T &aValue);
  /* pre:  openFiles has been successfully called.
      post: If a value has been successfully read, aValue
      holds that value and true is returned.
      Else, the read operation encountered an
      end-of-file and false is returned.  */


private:
      ... necessary members
};
```

Function readNextValue() should be implemented by the following guidelines:

- The function will read **one** values from the file and try to convert it to the type given by the actual type parameter `T`. The function will **not** handle the filtering of values outside the specified range.

- Error states in the current input stream will be handled internally in the function using `std::ios_base::failure` -exceptions from the stream-object and try / catch-block. If a read value cannot be converted, it should be written (as a string) to the file which opens with the parameter aErrorFile and to be analyzed later. After handling of an incorrect value, a new reading with an attempted of conversion should be done. This will be repeated until the function can return a value via output parameter aValue **or** end-of-file is encountered.

Filtering of the read values should be done with an instance of the class DataFilters:

```
template<typename T>
class DataFilter {
public:
      DataFilter(DataFileReader<T> *aReader, T aMin, T aMax);
      /* pre:  aReader points to an instance of DataFileReader<T>
            for which openFiles() has been succesfully called.
      */

      bool getNextValue(T &aValue);
      /* pre:  an earlier call to getNextValue() has not returned
            false.
         post: true is returned if aValue holds a value read from
            aReader. If a value could not be read, false
            is returned. If a value is read but is not within
            the interval specified by aMin and aMax parameters
            to the constructor, a range_error exception is
            thrown.
      */
private:
      ... necessary members
};
```

DataFilter <T> will be (via constructor) configured with a pointer to a DataFileReader <T> and min / max limits for valid data. `getNextValue()` uses the instance of DataFileReader <T> to retrieve the next value from the file. Values outside the specified range should result in a `std::range_error`-exception with a string representation of the unaccepted value as a parameter.

A test program will use classes described above to read the values from Values.dat and present the results.

- Accepted values will be within the interval [0.0 .. 10.0].
- Exceptions from the Data Filter should be managed in a way that all values outside the range are written to a file named `RangeErrors.dat`.

- The values which cannot be converted in the `DataFileReader` will be written to a file named `ReadErrors.dat`.

Requirements for solution:

- Stated pre- and post-conditions should be applied.
- Member function readNextValue should internally utilize exceptions from the input stream.
- The solution must give correct result:
  - numeric values: 19773
  - Value outside the interval : 201
  - Total: 99702.5
  - Average: 5.04235

Presentation

Zipped file with the class definitions + implementations + test program +Values.dat