Linköping University
Department of Computer and Information Science (IDA)
The UPP group

2015-09-16

Managing streams and errors

Aim

In this assignment you will learn how to use the standard stream types and how to handle stream errors.

Reading instructions

- Streams (ios, iostream)
  - The generic stream base class (istream, ostream)
  - The generic stream class instances (cin, cout, cerr, clog)
  - File streams (ifstream, ofstream, open(), close())
  - String streams (istringstream, ostringstream)
  - Stream errors (fail(), bad(), eof(), good(), clear())
- Command line arguments (argc, argv)
- Reading to end of input correctly
- Reading line by line with stringstreams
- (optional) Requesting and catching stream exceptions

Assignment

The game Trimino is a version of Domino, but with triangular bricks. We will not bother with the rules of the game, but rather focus our attention to the bricks used. Each of the three sides on a brick are labeled with a number. To be considered a correct Trimino brick those numbers have to fulfill a set of conditions:

1. The numbers have to be within a specified range. Different sets of bricks can have different range.
2. The numbers must be clockwise equal or increasing, starting from the smallest. If you place the brick with the smallest number down, then the largest number will be on the right side, and the number on the left side will be at least as large as the smallest and at most as large as the largest.

We show an example of the second condition in figure 1. Brick 2-5-3 is wrong, the numbers increase counterclockwise. If you place it with the smallest number (2) down the largest number (5) will be on the left side. Provided the range is at least [2-8] brick 8-2-5 in the figure is correct.

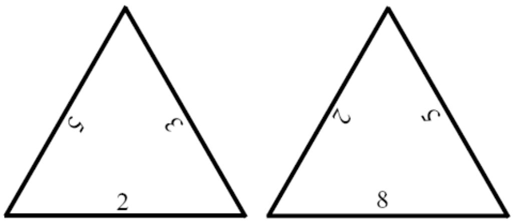

Figure 1: Brick 2-5-3 and 8-2-5

A computer game allows you to play Trimino using your custom set of bricks. The custom brick set is specified in a simple text file with three numbers on each line, and optionally the name of an image to show on the front of the brick. The file may look like this:

```
2 5 3  mosaic.png
1 4
8 2 5  C:\Application Data\Images\swirl.png
    7  12 53
6 9 4
flower.jpg
```

In the example brick set above line 1 contain a counterclockwise brick, line 2 is missing one number, lines 3-5 are promising but may be out of range and the last line is missing all three numbers.

Unfortunately the game does not provide a way to verify that a set is correct. You will write this verification program, and ensure that it works correctly.

Your program will open the file and check that each line contain a valid brick. You do not have to validate the optional image specification, and you can not make any assumptions about its format. Your final program have to get the file name and brick range from the command line. If the command line is empty (only program name specified) your program should ask for the necessary data interactively (until the user gets it right).

On any error the program should print detailed information about the error and what may have caused it on the standard error stream. The information should include brick information, file name and line number as relevant for the error in question.

Your program should further be implemented in a way that allow you to change (in your source code) a single input argument (and recompile) to read from standard input instead of from file. A more advanced solution interpret the file name `-` as standard input in order to let the user make the choice (runtime, without recompilation), but this solution is optional.

You are expected to create your own set of test files to ensure that your faulty brick detection behaves as specified, and that clear error information is shown in all situations.