MNXB01-ProjectNPT

Nikolei Höglinger
Pinar Öncel
Tilde Bonnevier Wallstedt

28 October 2022

1 Introduction

The goal of our project is to analyze the meteorological data that have been recorded by the Swedish Meteorological and Hydrological Institute (SMHI). As part of the SMHI OpenData Initiative, SMHI regularly publishes their datasets. The dataset that is used in our project contains the oldest recordings of average daily temperatures in Uppsala starting from the 12th January 1722.

The project work is divided into three parts: (1) **Reading the data** from the input file (2) **Analyzing the data** that have been read (3) **Plotting the data** that have been analyzed.

The project repository contains the main program source code file named **project.cpp** in its root directory along with other appropriate repository files such as *README.md*, *License*, *ChangeLog*, etc... Four *translation units* are created specifically for the project: (1) **Record** (2) **Read** (3) **Analyze** (4) **Plot**. *The header files* of the translation units can be found in the **include** directory and *the source code files* of the translation units can be found in the **src** directory.

2 Program execution

The execution of our program can be explained in three main steps: (1) **Read** step (2) **Analyze** step (3) **Plot** step.

Reading consists of reading from a file and storing all of the information in a vector of Record objects. The class **Record** is created specifically to store the existing six variables of each SMHI recording.

Analyzing consists of calling different functions to create vectors that contains only the specific data we would like to study further.

Plotting consists of utilizing the analysis output vectors to draw the graph objects that visualizes the specific data we would like to study.

3 Reading the data

For the reading of the data from the input files, a translation unit named **Read** is created, consisting of one header file named **Read.h** and one source code file named **Read.cpp**.

Another translation unit named **Record** is also created to store all six existing variables of each SMHI recording: year, month, day, temperature, urban temperature, and data ID number.

3.1 *cleandatafile.sh* shell script

cleandatafile.sh shell script cleans the input data file and writes the cleaned-up data into a new output file.

3.2 *readFromFile()* method

readFromFile() method reads the input data file using **filestream** *fstream* from the C++ standard library and returns *a vector of Record objects*.

3.3 *Record* class

Record class is used to instantiate new *Record objects* containing the data read from the input file.

4 Analyzing the data

For the analysis of the data that have been read from the input files in the previous step, a translation unit named **Analyze** is created, consisting of one header file named **Analyze.h** and one source code file named **Analyze.cpp**.

4.1 *Analyze.h* header file

The **Analyze.h** header file contains *the definitions of the methods* that are used in the different steps of analysis.

4.2 *Analyze.cpp* source code file

The **Analyze.cpp** source code file contains *the implementations of the methods* that are defined in its corresponding header file.

4.3 *Analyze* methods

4.3.1 *aveTempsPerYear()* method

aveTempsPerYear() method accepts *a vector of Record objects* and returns a vector of average temperatures of every year.

4.3.2 *diffTempsPerYear()* method

diffTempsPerYear() method accepts *a vector of Record objects* and returns a vector of temperature differences of every year, that is, the difference between the warmest and the coldest days of every year.

4.3.3 *getAllYears()* method

getAllYears() method accepts *a vector of Record objects* and returns a vector of all recorded years.

4.3.4 *maxTempsPerYear()* method

maxTempsPerYear() method accepts *a vector of Record objects* and returns a vector of max temperatures of every year, that is the warmest days of every year.

4.3.5 *minTempsPerYear()* method

minTempsPerYear() method accepts *a vector of Record objects* and returns a vector of min temperatures of every year, that is the coldest days of every year.

4.3.6 *nbrDaysPerMonth()* method

nbrDaysPerMonth() method accepts *a month of the year* and returns the number of days in the given month.

4.3.7 *tempsOnDay()* method

tempsOnDay() method accepts *a vector of Record objects* and *a day of the year*, and returns a vector of recorded temperatures on the given day.

4.3.8 *tempsPerDay()* method

tempsPerDay() method accepts *a vector of Record objects* and returns a vector of recorded temperatures on every day of the year.

4.3.9 *toArray()* method

toArray() method converts a *vector* into an *array.*

5 Plotting the data

For plotting, three functions were created in the Plot.cpp file (one for each analysis). Each of the functions takes the vectors of data as input, and creates graphs of them using TGraph.

The functions for the mean temperature each year and the temperature on Christmas every year takes 2 parameters: the vector of the years and the vector of the temperatures.

The function for plotting the maximum and minimum temperatures as well as the difference between them needed 4 parameters: 1 for the vector of years, and 3 for the 3 different vectors of temperatures (minimum, maximum, difference).

6 Results

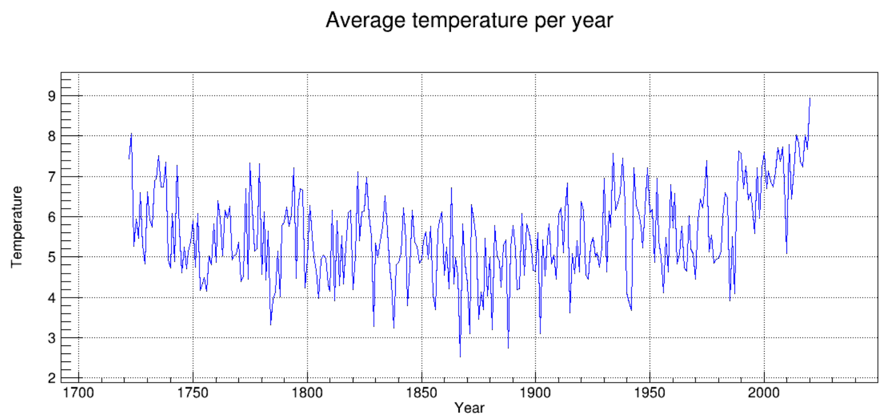

Figure 1: Average temperature of the year

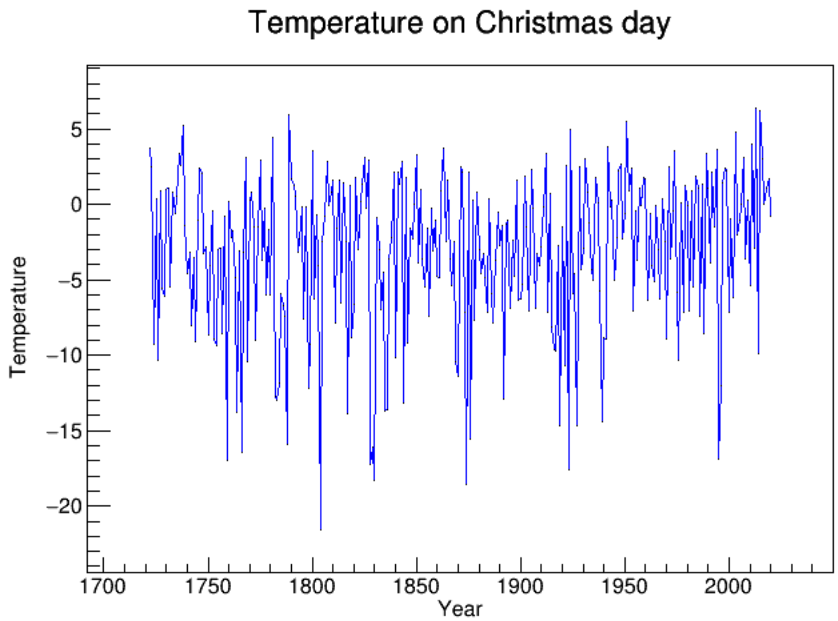

Figure 2: Temperature on Christmas

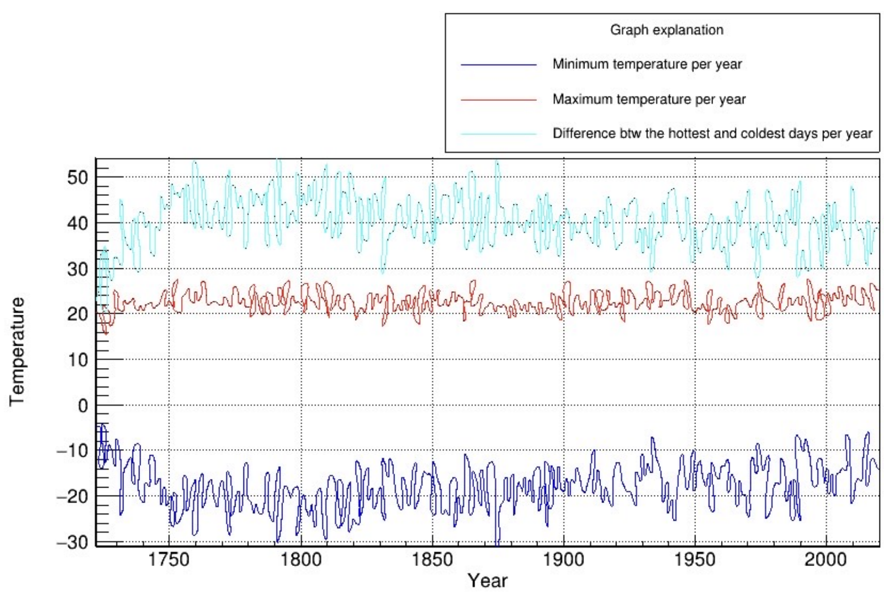

Figure 3: Minimum & maximum temperature and the difference between them

7 Discussion

The expectation was that the temperature would increase, due to climate change. However, all of the graphs appear rather flat. The graph of average temperature of the year appears to be increasing slightly, whereas the difference between the warmest and colest day of the year appears to decrease. The reason that a large increase in temperature is not observed might be that the data is taken from a rather short time range, so the effects are not clearly visible. Furthermore, small (but significant) changes in temperature are not easily observed.