MNXB01 Project NPT

Nikolei Höglinger
Pinar Öncel
Tilde Bonnevier Wallstedt

Lund University

28 October 2022

Short description of our team

- Nikolei: Clean and read data from input file
- Pinar: Analyze data that have been read
- Tilde: Plot data that have been analyzed

Our output

- Christmas day for every year
- Average temperature of the year for every year
- Coldest (minTemp) and hottest (maxTemp) of the year together with the difference between the two (diffTemp) for every year

Program execution

```cpp
void project() {

// ------------------------------------------------------------------------
// STEP 0 : program start
// ------------------------------------------------------------------------
    cout << " ... starting ProjectNPT ... " << endl;

// ------------------------------------------------------------------------
/* STEP 1 : READ -> by Nikolei
 * read from file and store data in a vector of Record objects */
// ------------------------------------------------------------------------
    string filename = "in/uppsala_tm_1722-2020.dat";

    vector<Record> records = readFromFile(filename);

// ------------------------------------------------------------------------
/* STEP 2 : ANALYZE -> by Pinar
 * analyze data and create vectors to be used in plotting */
// ------------------------------------------------------------------------
    vector<double> temps     = tempsOnDay(records, 12, 25);
    vector<double> years     = getAllYears(records);
    vector<double> aveTemps  = aveTempsPerYear(records);
    vector<double> minTemps  = minTempsPerYear(records);
    vector<double> maxTemps  = maxTempsPerYear(records);
    vector<double> diffTemps = diffTempsPerYear(records);
    vector<vector<double>> tempss = tempsPerDay(records);

// ------------------------------------------------------------------------
/* STEP 3 : PLOT -> by Tilde
 * plot and save graphs using the processed data */
// ------------------------------------------------------------------------
    make_graph_day(years, temps);
    make_graph_ave(years, aveTemps);
    make_graph_min_max_diff(years, minTemps, maxTemps, diffTemps);

// ------------------------------------------------------------------------
    // STEP 4 : program end
// ------------------------------------------------------------------------
    cout << " ... ending ProjectNPT ... " << endl;
}
```

Cleaning the data files - 1/2

smhi-opendata_1_53430_20210926_101122_Lund.csv

```
Stationsnamn;Klimatnummer;Mäthöjd (meter över marken)
Lund;53430;2.0

Parameternamn;Beskrivning;Enhet
Lufttemperatur;momentanvärde, 1 gång/tim;degree celsius

Tidsperiod (fr.o.m);Tidsperiod (t.o.m);Höjd (meter över havet);Latitud (decimalgrader);Longitud (decimalgrader)
1863-01-01 00:00:00;1974-05-31 23:59:59;73.0;55.7089;13.2026
1974-06-01 00:00:00;1992-12-06 23:59:59;50.0;55.7089;13.2026
1992-12-07 00:00:00;1997-05-31 23:59:59;25.0;55.6930;13.2290
1997-06-01 00:00:00;2021-09-01 06:00:00;26.451;55.6932;13.2251

Datum;Tid (UTC);Lufttemperatur;Kvalitet;;Tidsutsnitt:
1863-01-01;07:00:00;3.6;G;;Kvalitetskontrollerade historiska data (utom de senaste 3 mån)
1863-01-01;13:00:00;3.9;G;;Tidsperiod (fr.o.m.) = 1863-01-01 00:00:00 (UTC)
1863-01-01;20:00:00;4.5;G;;Tidsperiod (t.o.m.) = 2021-06-01 06:00:00 (UTC)
1863-01-02;07:00:00;4.6;G;;Samplingstid = Ej angivet
1863-01-02;13:00:00;3.7;G;;
1863-01-02;20:00:00;3.5;G;;Kvalitetskoderna:
1863-01-03;07:00:00;3.3;G;;Grön (G) = Kontrollerade och godkända värden.
1863-01-03;13:00:00;4.3;G;;Gul (Y) = Misstänkta eller aggregerade värden. Grovt kontrollerade arkivdata och
okontrollerade realtidsdata (senaste 2 tim).
1863-01-03;20:00:00;3.6;G;;
1863-01-04;07:00:00;2.3;G;;Orsaker till saknade data:
1863-01-04;13:00:00;3.4;G;;  stationen eller givaren har varit ur funktion.
1863-01-04;20:00:00;2.7;G
1863-01-05;07:00:00;1.5;G
1863-01-05;13:00:00;2.3;G
1863-01-05;20:00:00;0.9;G
1863-01-06;07:00:00;1.4;G
1863-01-06;13:00:00;2.2;G
1863-01-06;20:00:00;2.0;G
1863-01-07;07:00:00;2.2;G
1863-01-07;13:00:00;2.6;G
1863-01-07;20:00:00;2.9;G
1863-01-08;07:00:00;3.2;G
1863-01-08;13:00:00;3.4;G
1863-01-08;20:00:00;2.7;G
1863-01-09;07:00:00;1.3;G
1863-01-09;13:00:00;3.1;G
1863-01-09;20:00:00;2.5;G
1863-01-10;07:00:00;2.0;G
1863-01-10;13:00:00;2.0;G
1863-01-10;20:00:00;2.3;G
1863-01-11;07:00:00;1.7;G
```

```
[nikolei@aurora-rviz02 MNXB01-ProjectNPT]$ ./clean_datafile.sh smhi-opendata_1_53430_20210926_1011
22_Lund.csv
File smhi-opendata_1_53430_20210926_101122_Lund.csv found. Starting to clean data file.
Cleaning raw data set...

Cleaned data is stored in clean_smhi-opendata_1_53430_20210926_101122_Lund.csv.
[nikolei@aurora-rviz02 MNXB01-ProjectNPT]$ more clean_smhi-opendata_1_53430_20210926_101122_Lund.c
sv
2021-05-25 18:00:00 10.5 G
2021-05-26 06:00:00 9.8 G
2021-05-26 18:00:00 10.6 G
2021-05-27 06:00:00 10.2 G
2021-05-27 18:00:00 12.9 G
2021-05-28 06:00:00 11.8 G
2021-05-28 18:00:00 14.3 G
2021-05-29 06:00:00 11.2 G
2021-05-29 18:00:00 14.3 G
2021-05-30 06:00:00 13.1 G
2021-05-30 18:00:00 15.4 G
2021-05-31 06:00:00 13.3 G
2021-05-31 18:00:00 19.8 G
2021-06-01 06:00:00 14.0 G
[nikolei@aurora-rviz02 MNXB01-ProjectNPT]$
```

clean_datafile.sh

Cleaning the data files - 2/2

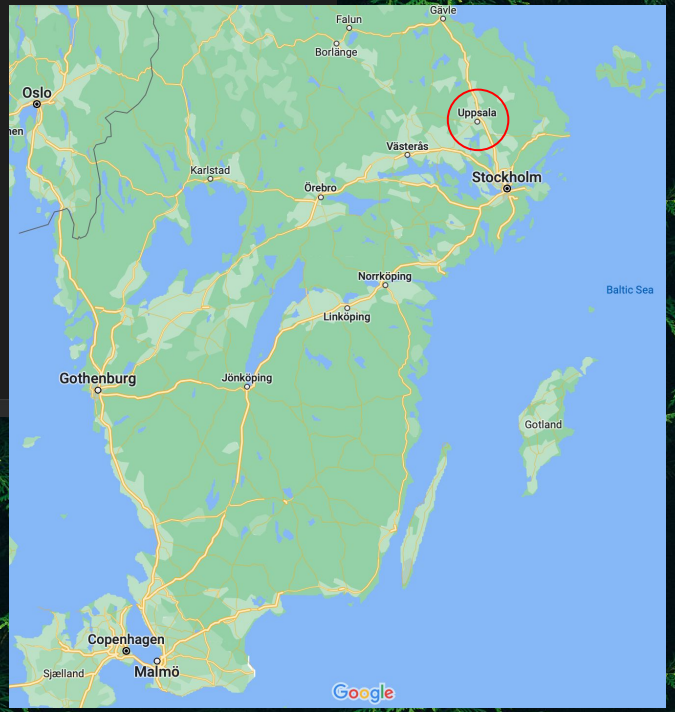
```bash
#!/bin/bash

#######################################################################
#
# Descritpion: This script takes as input
#               - A raw data file that contains first few lines of text and rows of data sets
#               - Creates a temporary file that will copy and process the origianl data file
#               - Copies processed temporary file into a cleaned data file.
#               - Removes temporary file
#
# Example on how to initialise script:
#               ./clean_data.sh <Data File Name>.csv
#
#######################################################################
# Functions. When error is present, one of these functions is called to explain how to use the script.

missingparam(){
        echo "Missing first parameter"
        echo "Usage:"
        echo "   $0 <Data File Name>.csv"
        echo
        echo "Exiting"
}

manyparam(){
        echo "Too many parameters given."
        echo "Usage:"
        echo "   $0 <Data File Name>.csv"
        echo
        echo "Exiting"
}

directoryfile(){
        echo "Input is a directory and not a file."
        echo "Usage:"
        echo "   $0 <Data File Name>.csv"
        echo
        echo "Exiting"
}
```

```bash
#######################################################################

DATAFILEINPUT=$1

if [[ $# == 0 ]]; then
        missingparam
        #exiting with error
        exit 1;
elif [[ $# -ge 2 ]]; then
        manyparam
        #exiting with error
        exit 1;
fi

DATAFILE=$(basename $DATAFILEINPUT)

if [ -f "$DATAFILE" ]; then
        echo "File $DATAFILE found."
        cp $DATAFILE clean_$DATAFILE

elif [ -d "$DATAFILE" ]; then
        directoryfile
        exit 1;
else
        echo "File not found."
        echo "Exiting"
        exit 1;
fi

LASTLINE=$(grep -n 'Datum' $DATAFILE | cut -d ":" -f 1) # Variable {LASTLINE} contains line number before data starts.
STARTDATA=$(($LASTLINE + 1)) # Variable {STARTDATA} contains line number where data starts.

# Manipulating clean_$DATAFILE file data set to the desired state
echo "Cleaning raw data set..."
tail -n -$STARTDATA $DATAFILE | cut -d ";" -f 1-4 | sed 's/;/ /g' > clean_$DATAFILE

echo
echo "Cleaned data is stored in clean_$DATAFILE."
```

Reading from the data files - 1/3

uppsala_tm_1722-2020.txt

```
1722  1 12   1.9   1.8 1
1722  1 13   2.3   2.2 1
1722  1 14   1.8   1.7 1
1722  1 15    .9    .8 1
1722  1 16  -1.8  -1.9 1
1722  1 17    .5    .4 1
1722  1 18    .1    .0 1
1722  1 19  -1.8  -1.9 1
1722  1 20    .5    .4 1
1722  1 21   1.8   1.6 1
1722  1 22   1.4   1.2 1
1722  1 23  -2.7  -2.9 1
1722  1 24   1.4   1.2 1
1722  1 25   1.8   1.6 1
1722  1 26   4.0   3.8 1
1722  1 27   4.0   3.8 1
1722  1 28   1.9   1.7 1
1722  1 29   3.2   2.9 1
1722  1 30   2.7   2.4 1
1722  1 31   1.3   1.0 1
1722  2  1  -1.3  -1.6 1
1722  2  2   1.4   1.1 1
1722  2  3   1.4   1.1 1
1722  2  4    .5    .2 1
1722  2  5   1.4   1.0 1
1722  2  6    .9    .5 1
1722  2  7   -.4   -.8 1
1722  2  8   -.4   -.8 1
1722  2  9  -1.4  -1.8 1
1722  2 10   2.3   1.9 1
```




```cpp
// project.cpp

#include <fstream>
#include <iostream>
#include <string>
#include <vector>

#include "include/Analyze.h"
#include "include/Plot.h"
#include "include/Read.h"
#include "include/Record.h"

using namespace std;

void project() {

// ------------------------------------------------------------------------
// STEP 0 : program start
// ------------------------------------------------------------------------
    cout << " ... starting ProjectNPT ... " << endl;

// ------------------------------------------------------------------------
/* STEP 1 : READ -> by Nikolei
 * read from file and store data in a vector of Record objects */
// ------------------------------------------------------------------------
    string filename = "in/uppsala_tm_1722-2020.dat";

    vector<Record> records = readFromFile(filename);

```

Up: project.cpp

Left: GeoBasis-DE/BKG (2009)

Reading from the data files - 2/3

```cpp
// Read.cpp by Nikolei

#include <fstream>
#include <iostream>
#include <string>
#include <vector>

#include "../include/Record.h"

// ---------------------------------------------------------------------------
// Returns a vector of Record objects storing the data from the input file
std::vector<Record> readFromFile(std::string filename) {

    std::vector<Record> records;

    std::ifstream input(filename);

        // Check if file is opened:
        if (!input) {
                throw std::runtime_error{"The file: " + filename + "could not be opened!"};
        }

    std::string line{};

    while(getline(input, line)) {
        int    year      = stoi(line.substr(0, 4));
        int    month     = stoi(line.substr(5, 3));
        int    day       = stoi(line.substr(8, 3));
        double temp      = stod(line.substr(11, 6));
        double tempUrban = stod(line.substr(17, 6));
        int    dataId    = stoi(line.substr(23, 2));

                Record record = Record(year, month, day, temp, tempUrban, dataId);

        records.push_back(record);
        }

    std::cout << "size of records: " << records.size() << endl;
    return records;
}
```

```cpp

// Read.h by Nikolei

#include <iostream>
#include <fstream>
#include <string>
#include <vector>

#include "Record.h"

// ---------------------------------------------------------------------------
// Reading the data file input and returns a vector of Record objects
std::vector<Record> readFromFile(std::string filename);
```

Up: Read.h

Left: Read.cpp

Record class - 3/3

stores all of the data read the input file

```cpp
// Record.h by Nikolei

#ifndef RECORD_H
#define RECORD_H

using namespace std;

// ------------------------------------------------------------
// Class definition for Record
You, 9 hours ago | 1 author (You)
class Record {

    // Private class members
    private:

        // Class attributes
        int    _year;
        int    _month;
        int    _day;
        double _temp;
        double _tempUrban;
        int    _dataId;

    // Public class members
    public:

        // Constructor method to instantiate a new Record object
        Record(
            int    year,
            int    month,
            int    day,
            double temp,
            double tempUrban,
            int    dataId
        );

        // Getter methods to call class attributes
        int    year()      const {return _year;}
        int    month()     const {return _month;}
        int    day()       const {return _day;}
        double temp()      const {return _temp;}
        double tempUrban() const {return _tempUrban;}
        int    dataId()    const {return _dataId;}

        // Prints the data stored in the Record object
        void print() const;
};
// ------------------------------------------------------------

#endif // RECORD_H
```

Record.cpp

```cpp
// Record.cpp by Nikolei

#include <iostream>

#include "../include/Record.h"

using namespace std;

// ------------------------------------------------------------
// Class implementation for Record

// Constructor method to instantiate a new Record object
Record::Record(
            int    year,
            int    month,
            int    day,
            double temp,
            double tempUrban,
            int    dataId) {

            _year      = year;
            _month     = month;
            _day       = day;
            _temp      = temp;
            _tempUrban = tempUrban;
            _dataId    = dataId;
            }

// Prints the data stored in the Record object
void Record::print() const {
    cout << "year: "      << year()      << "\t"
         << "month: "     << month()     << "\t"
         << "day: "       << day()       << "\t"
         << "temp: "      << temp()      << "\t"
         << "tempUrban: " << tempUrban() << "\t"
         << "dataId: "    << dataId()    << endl;
}
// ------------------------------------------------------------
```

Record.h

Analyze header file

- tempsOnDay()
- tempsPerDay()
- aveTempsPerYear()
- minTempsPerYear()

maxTempsPerYear()

diffTempsPerYear()

```
// Analyze.h by Pinar

#include <algorithm>
#include <iostream>
#include <string>
#include <vector>

#include "Record.h"

// ---------------------------------------------------------------------------
// Returns a vector of recorded temperatures on the given day
vector<double> tempsOnDay(const vector<Record>& records, int month, int day);

// ---------------------------------------------------------------------------
// Returns the nbr of days of the given month
int nbrDaysPerMonth(int month);

// ---------------------------------------------------------------------------
// Returns a vector of recorded temperatures on every day of the year
vector<vector<double>> tempsPerDay(const vector<Record>& records);

// ---------------------------------------------------------------------------
// Returns a vector of all recorded years
vector<double> getAllYears(const vector<Record>& records);

// ---------------------------------------------------------------------------
// Returns a vector of average temperatures of every year
vector<double> aveTempsPerYear(const vector<Record>& records);

// ---------------------------------------------------------------------------
// Returns a vector of min temperatures of every year = coldest days
vector<double> minTempsPerYear(const vector<Record>& records);

// ---------------------------------------------------------------------------
// Returns a vector of max temperatures of every year = hottest days
vector<double> maxTempsPerYear(const vector<Record>& records);

// ---------------------------------------------------------------------------
// Returns a vector of temperature differences of every year = the difference
between the hottest and the coldest day of every year
vector<double> diffTempsPerYear(const vector<Record>& records);

// ---------------------------------------------------------------------------
// Converts a vector into an array
// double* convertVtoA(const vector<double>& vector);
```

Analyze source code file : tempsOnDay() method

```
// ---------------------------------------------------------------------------
// Returns a vector of recorded temperatures on the given day
vector<double> tempsOnDay(const vector<Record>& records, int month, int day) {
    vector<double> temps;
    for (auto record : records) {
        if (record.month() == month && record.day() == day) {
            temps.push_back(record.temp());
        }
    }
    cout << "size of temps: " << temps.size() << endl;
    return temps;
}
```

Analyze source code file : tempsPerDay() method

```
// ---------------------------------------------------------------------------
// Returns the nbr of days of the given month
int nbrDaysPerMonth(int month) {
    if (month == 2) {
        return 28;
    }
    else if (month == 4 || month == 6 || month == 9 || month == 11) {
        return 30;
    }
    else {
        return 31;
    }
}

// ---------------------------------------------------------------------------
// Returns a vector of recorded temperatures on every day of the year
vector<vector<double>> tempsPerDay(const vector<Record>& records) {
    vector<vector<double>> tempss;
    for (int m = 1; m < 13; m++) {
        int nbrDays = nbrDaysPerMonth(m);
        for (int d = 1; d < nbrDays+1; d++) {
            vector<double> temps = tempsOnDay(records, m, d);
            tempss.push_back(temps);
        }
    }
    cout << "size of tempss: " << tempss.size() << endl;
    return tempss;
}
```

Analyze source code file : aveTemps PerYear() method

```
// ----------------------------------------------------------------------
// Returns a vector of average temperatures of every year
vector<double> aveTempsPerYear(const vector<Record>& records) {
    vector<double> means;
    int initialYear = records.front().year();
    int finalYear   = records.back().year();
    for (int y = initialYear; y < finalYear+1; y++) {
        double sum = 0;
        int nbrDays = 0;
        for (auto record : records) {
            if (y == record.year()) {
                sum += record.temp();
                nbrDays++;
            }
        }
        double mean = sum / nbrDays;
        means.push_back(mean);
    }
    cout << "size of means: " << means.size() << endl;
    return means;
}
```

Analyze source code file : min/maxTempsPerYear() methods

```
// ------------------------------------------------------------------
// Returns a vector of min temperatures of every year = coldest days
vector<double> minTempsPerYear(const vector<Record>& records) {
    vector<double> mins;
    int initialYear = records.front().year();
    int finalYear   = records.back().year();
    for (int y = initialYear; y < finalYear+1; y++) {
        double min = 0;
        for (auto record : records) {
            if (y == record.year()) {
                if (record.temp() < min) {
                    min = record.temp();
                }
            }
        }
        mins.push_back(min);
    }
    cout << "size of mins: " << mins.size() << endl;
    return mins;
}
```

```
// ------------------------------------------------------------------
// Returns a vector of max temperatures of every year = hottest days
vector<double> maxTempsPerYear(const vector<Record>& records) {
    vector<double> maxs;
    int initialYear = records.front().year();
    int finalYear   = records.back().year();
    for (int y = initialYear; y < finalYear+1; y++) {
        double max = 0;
        for (auto record : records) {
            if (y == record.year()) {
                if (record.temp() > max) {
                    max = record.temp();
                }
            }
        }
        maxs.push_back(max);
    }
    cout << "size of maxs: " << maxs.size() << endl;
    return maxs;
}
```

Analyze source code file : diffTemps PerYear() method

```
133 // ----------------------------------------------------------------------------
134 // Returns a vector of temperature differences of every year = the difference
    between the hottest and the coldest day of every year
135 vector<double> diffTempsPerYear(const vector<Record>& records) {
136     vector<double> diffs;
137     int initialYear = records.front().year();
138     int finalYear   = records.back().year();
139     for (int y = initialYear; y < finalYear+1; y++) {
140         double min = 0;
141         double max = 0;
142         for (auto record : records) {
143             if (y == record.year()) {
144                 if (record.temp() < min) {
145                     min = record.temp();
146                 }
147                 if (record.temp() > max) {
148                     max = record.temp();
149                 }
150             }
151         }
152         double diff = max - min;
153         diffs.push_back(diff);
154     }
155     cout << "size of diffs: " << diffs.size() << endl;
156     return diffs;
157 }
```

Plotting the data

- 3 functions were defined:
  - One for the average temperature every year
  - One for the temperature on Christmas every year
  - One for the maximum & minimum temperature every year and the difference
- They all take the vectors of data as input and create graphs from them using TGraph

Code for plotting

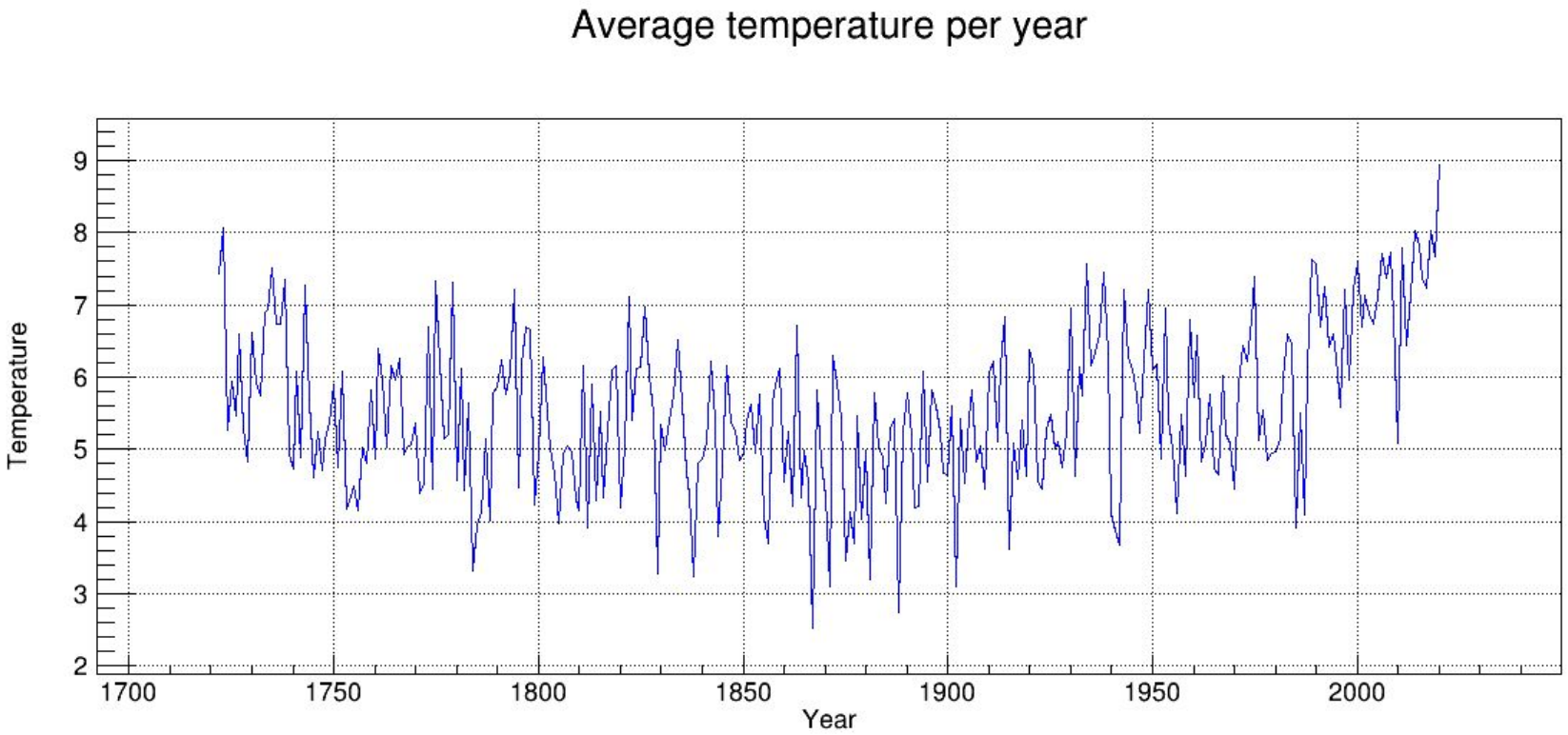
Here is one of the 3 functions that were made (the function for plotting the average temperature on Christmas day.

The other 2 functions did the same thing with minor tweaks.

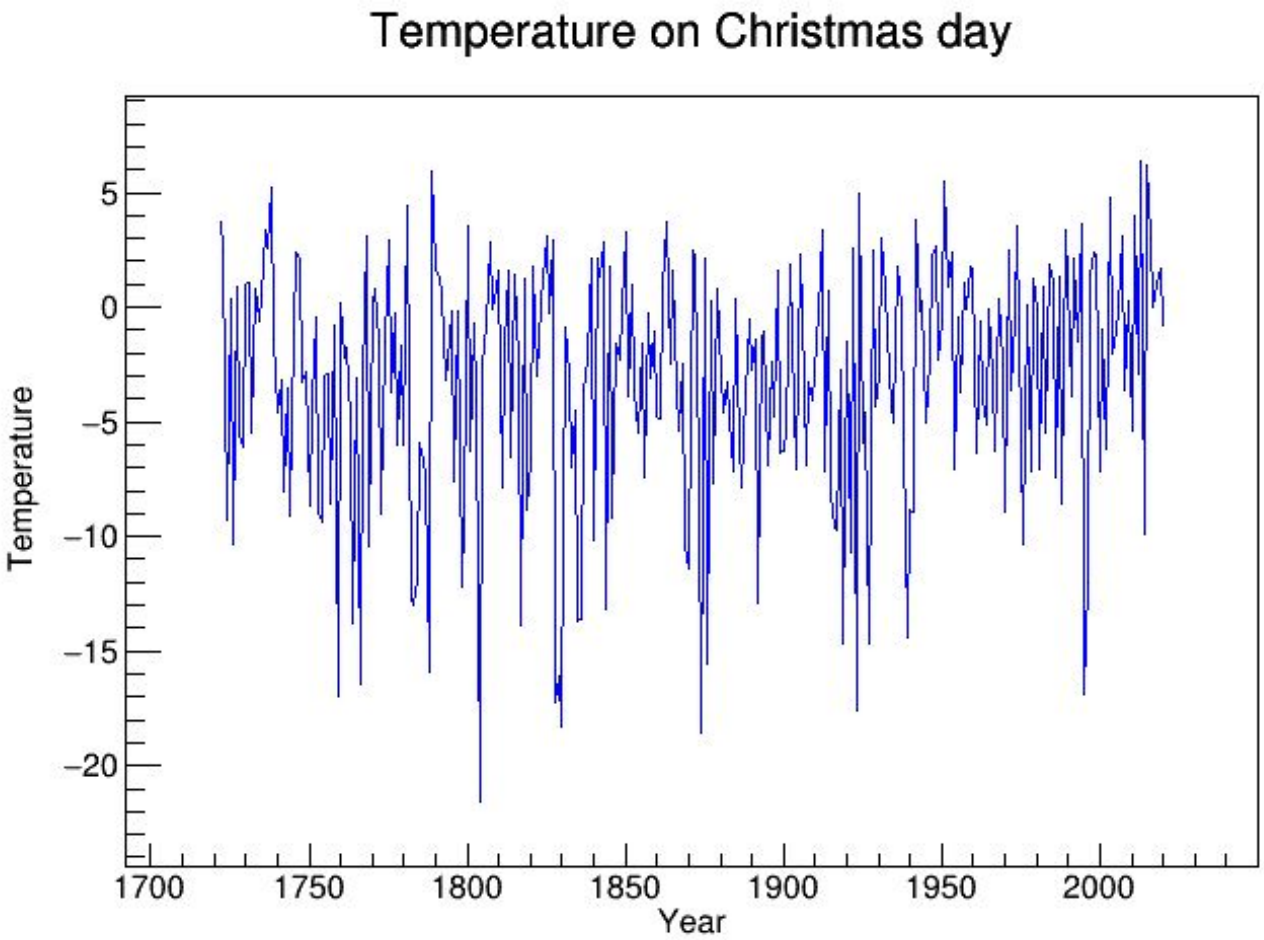
```
void make_graph_day(const vector<double>& years, const vector<double>& temps) {

    auto c_xmas = new TCanvas("c", "canvas for g");
    TGraph *g = new TGraph(years.size(), &(years[0]), &(temps[0]));

    g->SetTitle("Temperature on Christmas day");
    g->SetLineColor(4);

    g->GetXaxis()->SetTitle("Year");
    g->GetYaxis()->SetTitle("Temperature");
    g->GetXaxis()->CenterTitle(true);
    g->GetYaxis()->CenterTitle(true);

    gStyle->SetCanvasColor(0);
    gStyle->SetPadTopMargin(0.15);
    gStyle->SetPadBottomMargin(0.15);
    gStyle->SetPadLeftMargin(0.10);
    gStyle->SetPadRightMargin(0.10);
    gStyle->SetFrameFillColor(0);
    gStyle->SetPadGridX(true);
    gStyle->SetPadGridY(true);
    gStyle->SetPadColor(0);
    gStyle->SetCanvasDefW(1280);
    gStyle->SetCanvasDefH(720);

    g->Draw();

    c_xmas->SaveAs("out/graph_christmas.png");

    cout << "graph_christmas done" << endl;
}
```

Results

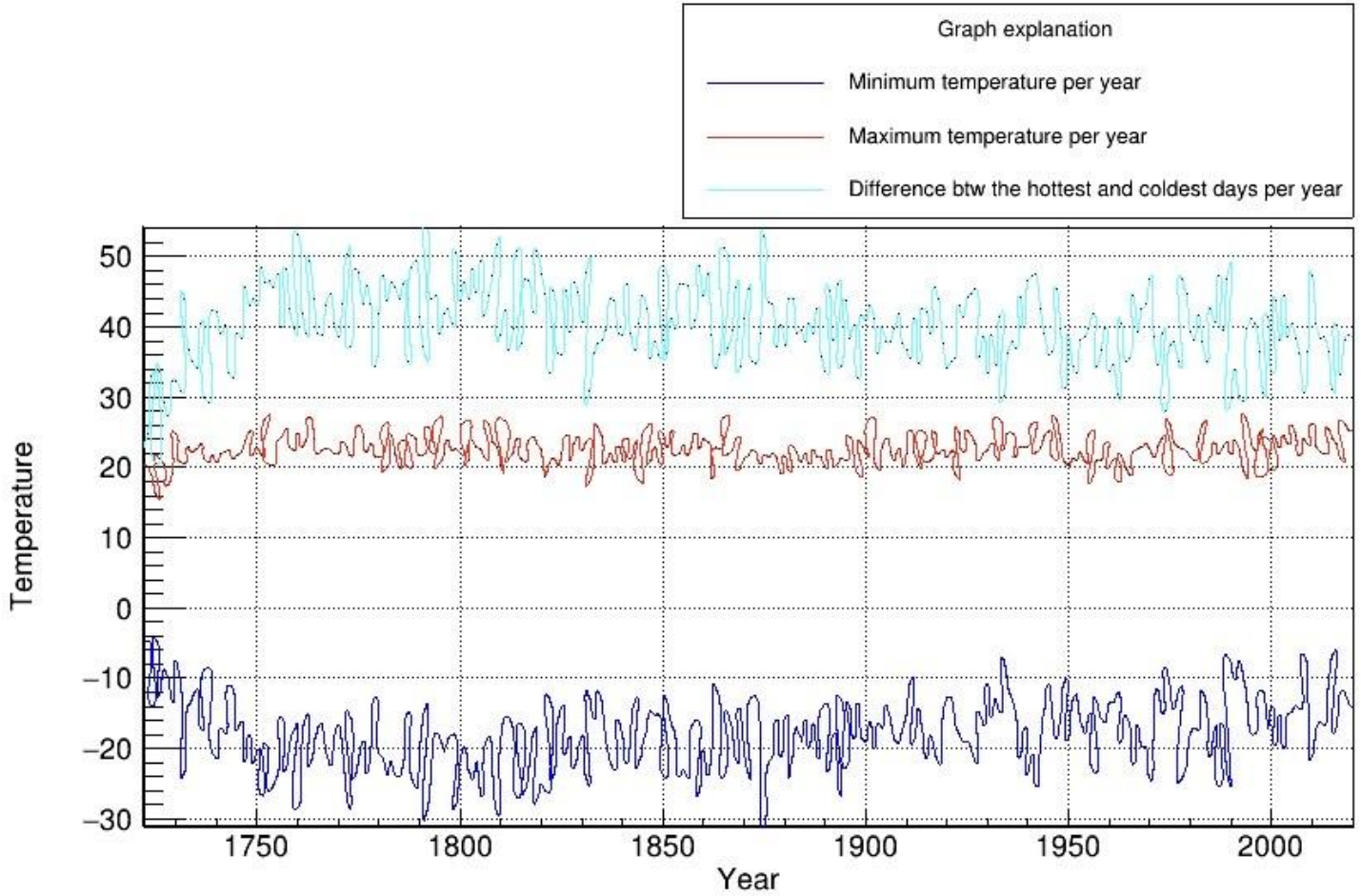
Average temperature of the whole year



Temperature on Christmas



Max & min temp. and difference between them



Thank you for listening!

Nikolei Höglinger
Pinar Öncel
Tilde Bonnevier Wallstedt

Lund University

28 October 2022