MNXB01Project

1 Introduction

1.1 Motivation

The goal of this project has been to read, manipulate and present large amounts of data with Bash, C++ and root tools at our disposal. More specifically, we are given a collection of temperature measurements spanning over several years and performed in different cities in Sweden, and asked to present it in an intuitive manner via graphs and histograms. The data which needs to be manipulated, has the format 'Date(yyyy-mm-dd);Time(hh:mm:ss);Temperature;Grade(G/Y)', and contains some other useful information, e.g city and measurement period, at the top.

1.2 Work distribution

The work load was distributed as following: Each group member was tasked with reconstructing one of the given examples. Ashar looked into the average daily temperature over a period of several years, and acted as the release manager. Zack focused on presenting data for the warmest and coldest day of each year in a given time period. Finally, Jesper looked into the temperature of a given day in the year. In addition, Jesper also created a histogram comparing the temperature over a month between two data dets.

Although our tasks require very similiar methods, we all took different paths in constructing our results. Firstly, Jesper took the "traditional" way of writing a pure root script, loading it in root and calling the relevant function from there. Zack on the other hand, went in the direction of compiling C++ code outside of a root session, instead providing root flags in the compilation. Lastly, Ashar wrote a C++ class, but let root do the compilation. This served as an interesting example of how to access root in different ways.

2 Method

2.1 Work flow

Before moving onto the shared repository and implementation of the code, a few words should be said regarding the work flow. Due to the nature of the problem, each project could be developed independently, (for the most) in separate files. This allowed for simple merging, only producing slight conflicts in common files such as the README, WorkPlan, Rootlogon etc. Changes were committed by the independent developers often, with significant comments. At the end, an effort was made to "collect" the individual projects so that a new user can more easily construct the histograms, this is explained in the following section.

2.2 Repository environment

Once the final merge was complete, a (very slightly filtered) git log was placed in the ChangeLog, including commit hash, comment, author, date and files affected. In the same directory, a README file can be found, explaining to the user how to construct each histogram. In the case of Ashar's (TempPerDay) and Jesper's (TempOnDay and Compare) histograms, which are constructed inside root, a shared project.cpp file was created. This file contains functions, which when called, themselves call on the appropriate files to be called in root, to create the desired histograms. Entering root, the projet.cpp file is automatically loaded (in addition to other relevant files, such files are mentioned in the implementation part) by rootlogon.C. Finally, calling the above mentioned functions with appropriate input parameters (implementation section below mentions associated input parameter, if such exist for that histogram). Also, in the roologon.C file, we have defined plot traits such as background colour, so that the histograms have similar styles.

2.3 The temperature for every day over a period of years (TempPerDay)

The goal here is to plot the average daily temperature for every day, over some period of time. In this case, it is for Lund between 1981-1990. For the filtering process, a batch file (datacleaner2.sh) was created. First, it runs a for loop over the years we want the histogram to cover, and uses 'grep' to pick data entries (rows) belonging to those years. Using the append '>>' command, these entries can be appended to the desired file. With 'awk', one can then pick out the desired columns, in my case that was date and temperature. With the "inverse" 'grep', i.e 'grep -v 02-29', one can pick any entries *NOT* including 29 February, i.e exclude extra data from leap years. One can in principle keep these measurements, but the histogram must then contain 366 days, with 29 February only having contributions once every four years.

The second part consisted of writing C++ code in a class called tempTrender. The class code begins with a constructor, which takes the (const, since we will not be altering it inside the code) data file path as input, does some further manipulation of the data and stores the desired data into the member variables Temp and Error. Temp contains the average daily temperatures and

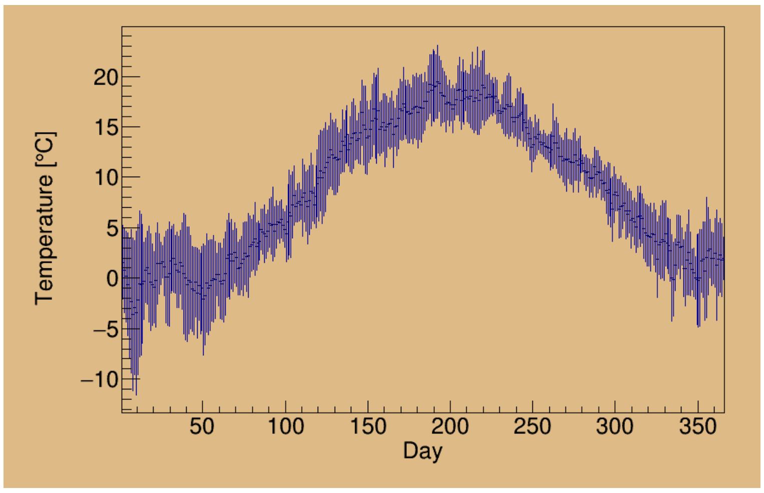

Figure 1: Average daily temperature in Lund between 1981-1990. 29 February from leap years has been omitted.

Error contains the associated errors of averaging. These are stored specifically as member variables so that they do not go out of scope once leaving the constructor. The constructor is declared in include/tempTrender.h header by

```
explicit tempTrender(const string& filePath
```

where explicit is added to avoid implicit conversions. The initialization is done in the src/tempTrender.cpp source file via

```
tempTrender::tempTrender(const string& filePath) {
   ...
   while (the file can be read in the format 'double >> int >> double') {
      [store measurements]
      if (3 rows/measurments have passed) {
         [Average over last three measurements to get daily average]
      }
   }
   for (every day of the year) {
      [Average  measurement of given day for every year and calculate associated error]
   }
}
```

The member variables are then read by the member function TempPerDay, which constructs the histogram. Similarly to the constructor, it is declared in the header and initialized in the source file. The function constructs bin content with associated errors using the Root built-in commands

```
perDayHist->SetBinContent(bin,Temp[bin]-1)
```

```
perDayHist->SetBinErro(bin,Error[bin-1])
```

inside a for loop running from bin=1→365. Note that in the code, the perDayHist belongs to the TH1D* class, meaning it is a pointer and to access its "value", we have to use the "deference and access" operator '$->$'. Finally, entering a root session and typing "TempPerDay()" (function defined in project.cpp calling the class), we are presented with Figure 1.

2.4 The temperature of a given day (TempOnDay)

We want a function that takes a month and day as argument and provides us with the temperature of that given day throughout the years. The first step is to filter the data, which was done through a batch file (data_cleaner.sh). By using 'awk', we pick the columns containing the date and temperature, and we disregard rows containing compromised data.

The second step was done in root. This function takes the argument of a date, then it loops through the filtered data set, if we encounter the matching day whilst looping through the data set, we fill the histogram with the temperature of that day. The third step is making sure that we can call the function from tempOnDay.C from project.cpp, and to do that, we do

```
#include "tempOnDay.C"
void TempOnDay(int Month, int Day) {
    system("data_cleaner.sh");
    tempOnDay(Month, Day);
}
...
```

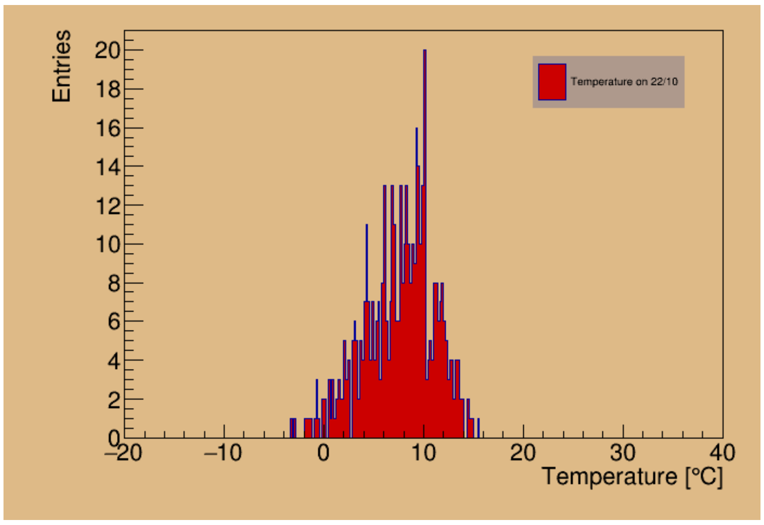

Figure 2: The temperature on 22nd October in Lund excluding data with grade Y.

and an example histogram can be seen in Fig. 2.

2.5 Comparing the temperature of a month between two data sets (Compare)

This was constructed in a similar way to TempOnDay. It uses the same batch file to filter the data. In the second step, it is the same idea, comparing the date with the input argument, but it is slightly differently implemented since we want the entire month and not just a day. In this step the histogram is filled. Then we can call the function from project.cpp similar to TempOnDay

```
#include "comp.C"
void Compare(int Month) {
    system("data_cleaner.sh");
    comp(Month);
}
...
```

and an example histogram can be seen in Fig. 3.

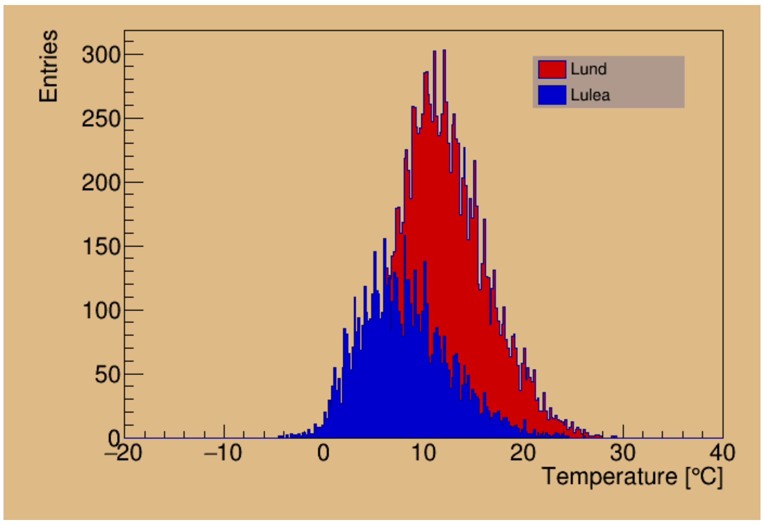

Figure 3: The temperature of May in Lund and Luleå excluding data with grade Y.

2.6 The warmest and coldest day of the year

Here for a given dataset the warmest and coldest days are plotted in the same histogram to show when the coldest and warmest days have occurred during the last few hundred years.

This was performed by first parsing the data using a bash script, datafilter.sh. This script ordered the weather for each year from coldest to warmest by use of the 'sort' command. The script would single out each year using the 'grep' command which would search through the dataset to isolate each year before performing the sort.

The next step would be to remove the coldest and hottest days of each year, which would be the last and first term of each year after the sorting, using the 'head' and 'tail' command. The two entries would then be stored in two different files, OUT_MIN.txt and OUT_MAX.txt respectively.

Finally the bash script converts the dates of the entries which are given in standard format, YYYY-MM-DD, into the day of year integer using search and replace command, 'sed'

```
# Get Max and Min for each year
for YEAR in {1884..2021}; do
    grep $YEAR $FILE-clean.csv | sort -k3 -g -t ';' | { head -n1 >>
    $OUT_MIN; tail -n1 >> $OUT_MAX; } > /dev/null;
done

# Change date to day of year
sed -i -r 's#([0-9]{4}-[0-9]{2}-[0-9]{2})(.*)#printf "%s%s" $(date -d "\1" +%j)
"\2";#ge' $OUT_MIN
```

The next step involved a c++ code that isolates the first column of the dataset, the day of year, which was needed to be perform the plot. Using 'ifstream' together with the 'getline' function, a simple while loop stores the first column into an output file, maxVals.txt minVals.txt.

```
    while(getline(inputfile, line)){
        string token = line.substr(0, line.find(';'));
        day = atoi(token.c_str());
        outfile << day << '\n';
        dys.push_back(day);
    }
```

With the cleaned up data now achieved the final step was to plot the histogram. This was again written in c++ code, but compiled using root config, with the following bash command. Inside the bash commands are flags to help locate the root config library.

```
    g++ hist1.cpp $(root-config --glibs --cflags --libs) -o main
```

Within the hist.cpp code a simple function plots the histogram is plotted using the root class 'THI1'.

```
void histo_plot(string input_name) {
     histo = new TH1I("hist", "", 365, 1, 365);
    file.open(input_name, ios::in);

    int value;
    while(!file.eof())
    {
    file >> value;
    histo->Fill(value);
    }

    histo->GetXaxis()->SetTitle("Day of Year");
    histo->GetYaxis()->SetTitle("Entries");

    histo->Draw("SAME");
    file.close();
}
```

This produced a simple histogram with the hottest and coldest days of every year since 1884. To take it one step a further a gaussian was fitted to the plot, and an extra set of code was written into the hist.cpp code, which was given in the project_instructions.pdf file, this time using the 'TF1' root class. The final plot is shown in Fig.4.

```
double Gaussian(double* x, double* par) { //A custom function
    return par[0]*exp(-0.5*(x[0]*x[0] - 2*x[0]*par[1] + par[1]*par[1])/(par[2]*par[2]));
}


 TF1* func = new TF1("Gaussian", Gaussian, 90, 300, 3);
func->SetParameters(5, 190, 50); //Starting values for fitting
func->SetLineColor(kBlack);
histo->Fit(func, "QR");
```

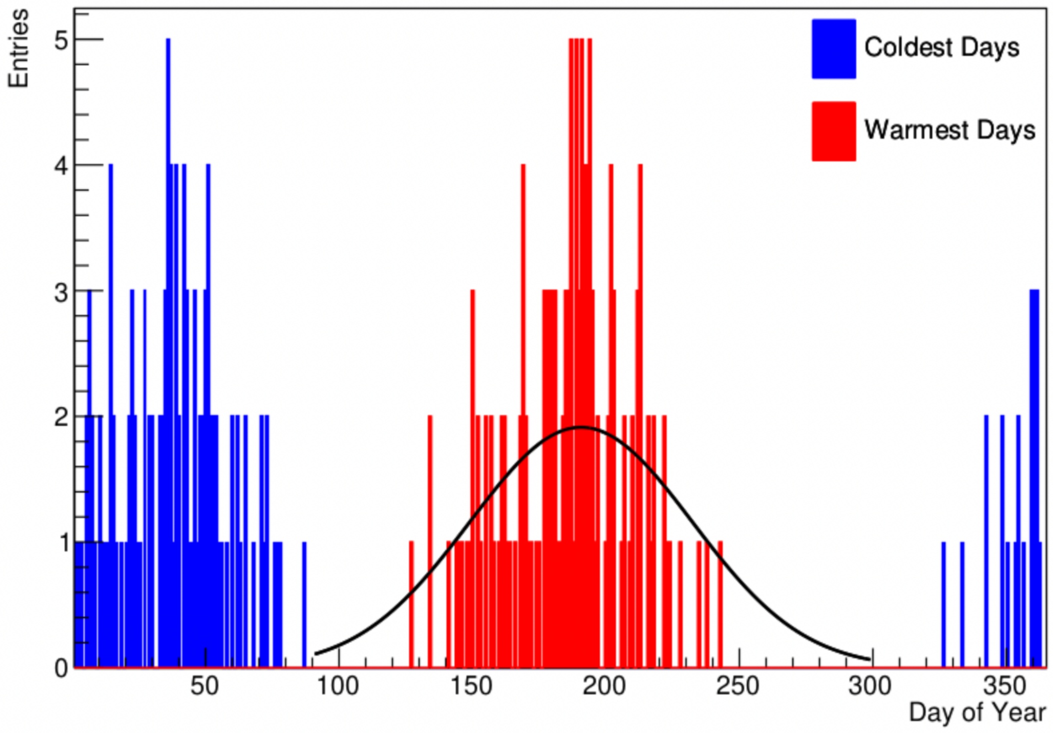

Figure 4: Hottest and coldest day of the year for Borås from 1884 to 2021

3 Conclusion

This assignment has certainly proven how far one can come ”only” using bash, C++ and Root when dealing with large sets of data. Not only does each one of these languages offer very powerful tools to manipulate large amounts of data in an efficient way, a problem can often be solved by more than one them, allowing the user a variety of options to tackle his or her problem. Obviously, each method has its own strong suit, e.g bash is very efficient at cutting and filtering text files compared to the other two, while Root gives a wider range of options for plots and histograms compared to C++. But most impressive of all is the simplicity in connecting the methods together to get a very natural and straightforward work flow. And of course, git should also be mentioned, allowing users to conveniently and safely work on group projects.