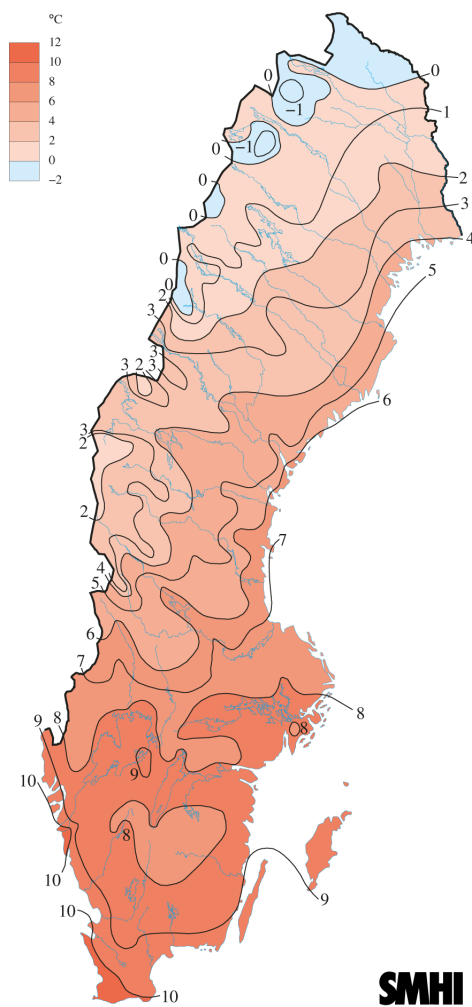


---

## Swedish climate study

---



# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Uppsala's climate study</b>	<b>3</b>
2.1	Mean temperature per year in Uppsala . . . . .	5
2.2	Temperature on a given day in Uppsala . . . . .	9
2.3	Hottest and coldest temperatures per year . . . . .	12
<b>3</b>	<b>Temperatures per days in Lund, Umea and Falsterbo</b>	<b>17</b>

# Chapter 1

## Introduction

The goal of this project is to study Swedish climate via different programs. We have access to various data from the Swedish Meteorological and Hydrological Institute (SMHI), containing the temperatures at different times, for a bunch of Swedish cities. We will be focusing on Lund, Uppsala, Umea and Falsterbo. We were given some examples of what kind of programs would fit the subject and we decided to code three of the examples and make three up by ourselves. Then we have assigned one or several code to write to each of use and put it in the workplan as following :

### Workplan

---

We to have select one of the available SMHI datasets and write a program that extracts information from the Swedish climate data.

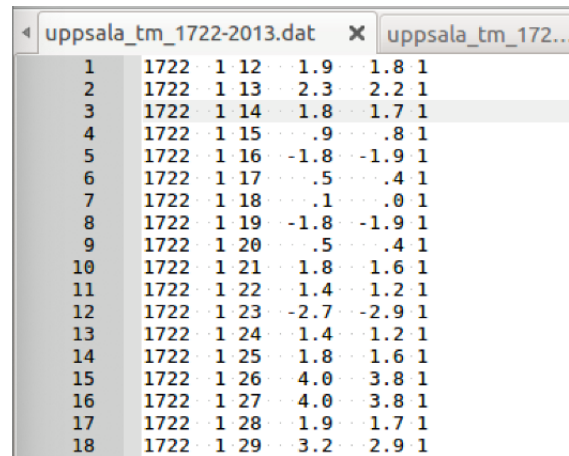
---

- Use ROOT to make plots and eventual statistical analysis: a small code skeleton
- Section 3 contains a few examples of what kind of information you could get from the data. Decide on at least three interesting results to produce.
  - Estelle - 3.4 example "The mean temperature of each year"
  - Johan - 3.1 example "The temperature of a given day"
  - Jasmain - "The hottest temperature per year"
  - Philip - "Temperature per year at a given time in a given town"
- To "clean" the input dataset we will use any of the tools we've learnt during the course (ROOT, C++, bash commands, bash scripts) or anything else, as long as we document the way we did and we do not modify the original data, the cleaning process is described and the cleaned up dataset must be a separate file or set of files.

## Chapter 2

# Uppsala's climate study

The data sheet used in this chapter consist in air temperature measurements from Uppsala in Sweden. The file format containing the data is .dat. Another file containing the description on datasets is in .txt. Figure 2.1 show a screenshot of the file associated with Uppsala temperature and figure 2.2 show a screenshot of the descritpion of the file associated with Uppsala data.



1	1722	1	12	1.9	1.8	1
2	1722	1	13	2.3	2.2	1
3	1722	1	14	1.8	1.7	1
4	1722	1	15	.9	.8	1
5	1722	1	16	-1.8	-1.9	1
6	1722	1	17	.5	.4	1
7	1722	1	18	.1	.0	1
8	1722	1	19	-1.8	-1.9	1
9	1722	1	20	.5	.4	1
10	1722	1	21	1.8	1.6	1
11	1722	1	22	1.4	1.2	1
12	1722	1	23	-2.7	-2.9	1
13	1722	1	24	1.4	1.2	1
14	1722	1	25	1.8	1.6	1
15	1722	1	26	4.0	3.8	1
16	1722	1	27	4.0	3.8	1
17	1722	1	28	1.9	1.7	1
18	1722	1	29	3.2	2.9	1

Figure 2.1: The raw data in Uppsala-tm-1722-2013.dat

First, I clean the file by writing a temporary code which replace the irregular space groups with a single space separating the data. This script will extract the data and save it to a new data file called tempdata\_uppsala.txt. The result is shown in Figure 2.3.

```

1 -----
2 NB! Users of the file 'uppsala_tm_1722_2013.dat'
3 are asked to refer to:
4 Bergström, H., Moberg, A.:
5 Daily air temperature and pressure series for Uppsala (1722-1998),
6 Climate Change, 53:213-252.
7 -----
8
9 The file 'uppsala_tm_1722_2013.dat' contains:
10 Daily temperature data for Uppsala 1722-2013.
11
12
13 column.....data
14 1-3 ..... Year, month, day
15 4 ..... Daily average temperature according to observations.
16 ..... Unit: °C
17 5 ..... Daily average temperatures corrected for the urban effect.
18 6 ..... Data id no. meaning data from:
19 ..... 1=Uppsala, 2=Risinge, 3=Betna, 4=Linköping, 5=Stockholm, 6=Interpolated
20 .....
21
22 Uppsala 2013-01-17
23 hans.bergstrom@met.uu.se

```

Figure 2.2: Description of Uppsala data file

```

tempdata_uppsala.txt
File Edit Search Options Help
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

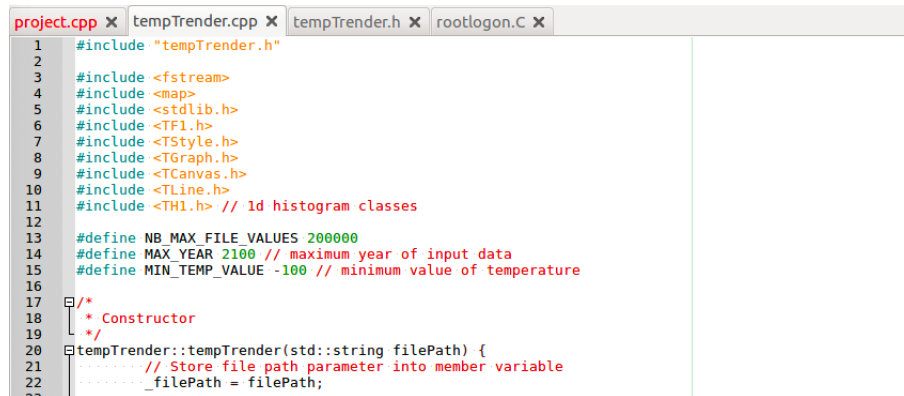
```

Figure 2.3: New file, tempdata\_uppsala.txt

## 2.1 Mean temperature per year in Uppsala

I will use the year of the file (first column in figure 2.3) and daily average temperatures (corrected by the urban effect, penultimate column), in order to show a histogram plotting average values of temperatures in Uppsala each year from 1722 to 2013.

I create a function that allow us to read this file and retrieve all the temperatures per year in order to calculate the mean temperature per year. So, I extract the first 4 characters, which corresponds to the year, then the 5th field which corresponds to the temperature. I sum the values as seen below in figures 2.4 and 2.5.

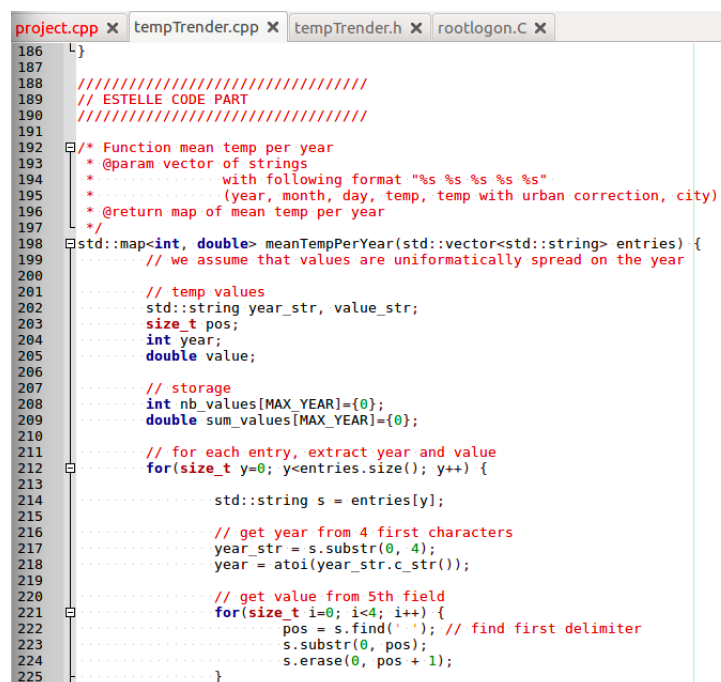


```

1  #include "tempTrender.h"
2
3  #include <fstream>
4  #include <map>
5  #include <stdlib.h>
6  #include <TF1.h>
7  #include <TStyle.h>
8  #include <TGraph.h>
9  #include <TCanvas.h>
10 #include <TLine.h>
11 #include <TH1.h> // 1d histogram classes
12
13 #define NB_MAX_FILE_VALUES 200000
14 #define MAX_YEAR 2100 // maximum year of input data
15 #define MIN_TEMP_VALUE -100 // minimum value of temperature
16
17 /*
18  * Constructor
19  */
20 tempTrender::tempTrender(std::string filePath) {
21     // Store file path parameter into member variable
22     _filePath = filePath;
23 }

```

Figure 2.4: Extract (a) from tempTrender.cpp



```

186 }
187
188 // ESTELLE CODE PART
189 // ESTELLE CODE PART
190
191 /*
192  * Function mean temp per year
193  * @param vector of strings
194  * with following format "%s %s %s %s %s"
195  * (year, month, day, temp, temp with urban correction, city)
196  * @return map of mean temp per year
197  */
198 std::map<int, double> meanTempPerYear(std::vector<std::string> entries) {
199     // we assume that values are uniformly spread on the year
200
201     // temp values
202     std::string year_str, value_str;
203     size_t pos;
204     int year;
205     double value;
206
207     // storage
208     int nb_values[MAX_YEAR]={0};
209     double sum_values[MAX_YEAR]={0};
210
211     // for each entry, extract year and value
212     for(size_t y=0; y<entries.size(); y++) {
213         std::string s = entries[y];
214
215         // get year from 4 first characters
216         year_str = s.substr(0, 4);
217         year = atoi(year_str.c_str());
218
219         // get value from 5th field
220         for(size_t i=0; i<4; i++) {
221             pos = s.find(' '); // find first delimiter
222             s.substr(0, pos);
223             s.erase(0, pos + 1);
224         }
225     }

```

Figure 2.5: Extract (b) from tempTrender.cpp

Then I calculate the mean temperature per year and the mean temperature for all the years. And finally I create a canvas object and draw the histogram, as you can see in extracts (d) and (e).

```
project.cpp x tempTrender.cpp x tempTrender.h x rootlogon.C x
225 .....}
226 .....pos = s.find('.');
227 .....value_str = s.substr(0, pos);
228 .....value = atof(value_str.c_str());
229 .....
230 .....// add value to arrays
231 .....nb_values[year]++;
232 .....sum_values[year] += value;
233 .....}
234 .....
235 .....// return variable
236 .....std::map<int, double> ret;
237 .....double mean;
238 .....
239 .....// for each year, compute mean value
240 .....for(size_t i=0; i<MAX_YEAR; i++) {
241 .....    if(nb_values[i]>0) {
242 .....        mean = sum_values[i]/nb_values[i];
243 .....        std::cerr << "year: " << i << " mean: " << mean << std::endl;
244 .....        ret.insert(std::make_pair<int, double>((int)i, (double)mean));
245 .....    }
246 .....}
247 .....
248 .....return ret;
249 .....}
250 .....
251 void tempTrender::tempPerYear(int yearToExtrapolate) {
252 .....// open input file
253 .....std::string line;
254 .....ifstream inputfile(_filePath.c_str());
255 .....
256 .....// create array to store values
257 .....std::vector<std::string> entries;
258 .....
259 .....// read values and store it into array, hist or something else
260 .....if(inputfile.is_open()) {
261 .....    while(getline(inputfile, line)) {
262 .....        entries.push_back(line);
263 .....    }
264 .....}
```

Figure 2.6: Extract (c) from tempTrender.cpp

After compilation, we are able to plot a histogram of mean temperature each year and the mean of all years, from 1722 to 2013.

```

project.cpp x tempTrender.cpp x tempTrender.h x rootlogon.C x
264 .....}
265 .....}
266 .....
267 .....// compute mean per year
268 .....std::map<int, double> meanPerYear = meanTempPerYear(entries);
269 .....
270 .....// compute mean for all time
271 .....double meanAllTime = 0;
272 .....for( std::map<int, double>::iterator it = meanPerYear.begin();
273 .....it != meanPerYear.end();
274 .....it++ ) {
275 .....    meanAllTime += it->second;
276 .....}
277 .....meanAllTime /= meanPerYear.size();
278 .....
279 .....std::cout << "meanAllTime = " << meanAllTime << endl;
280 .....
281 .....// create canvas for graph
282 .....TCanvas *c1 = new TCanvas("Estelle", "Project : Mean Temp Per Year");
283 .....
284 .....// create new histogram object
285 .....TH1F* hist = new TH1F("graph", "Mean Temp Per Year", meanPerYear.size(), 1722, 2100);
286 .....
287 .....// fill hist with mean temp per year values from input file
288 .....for( std::map<int, double>::iterator it = meanPerYear.begin();
289 .....it != meanPerYear.end();
290 .....it++ ) {
291 .....    // fill hist with date and value from mean temp per year map
292 .....    //std::cerr << "value : " << it->second << std::endl;
293 .....    hist->Fill(it->first, it->second);
294 .....}
295 .....
296 .....// Draw horizontal mean
297 .....TLine *meanline = new TLine (1722,meanAllTime,2013,meanAllTime);
298 .....
299 .....// This code is given from project instruction for creating the graph
300 .....TGraph* graph = new TGraph();
301 .....
302 .....//for(int bin = 1; bin < hist->GetNbinsX(); ++bin) {
303 .....

```

Figure 2.7: Extract (d) from tempTrender.cpp

```

304 .....// graph->Expand(graph->GetN() + 1, 100);
305 .....// graph->SetPoint(graph->GetN(), hist->GetBinCenter(bin),
306 .....// hist->GetBinContent(bin));
307 .....//}
308 .....//graph->Draw("SAME C");
309 .....
310 .....// create function for extrapolation
311 .....//TF1 *f = (TF1*)hist->GetFunction("pol7");
312 .....//f->Eval(yearToExtrapolate);
313 .....
314 .....//Axis title
315 .....hist->SetTitle("Mean temperature per year (Uppsala, 1722-2013)");
316 .....hist->GetXaxis()->SetTitle("Year");
317 .....hist->GetXaxis()->CenterTitle();
318 .....hist->GetYaxis()->SetTitle("Mean temperature (Celcius Deg)");
319 .....hist->GetYaxis()->CenterTitle();
320 .....
321 .....// draw hist
322 .....hist->Draw("SAME");
323 .....
324 .....// draw mean line
325 .....meanline->Draw();
326 .....
327 .....}
328 .....
329 .....

```

Figure 2.8: Extract (e) from tempTrender.cpp



```

courseuser@Lubuntu-VirtualBox: ~/MNXB01-Projet-2019/code
File Edit Tabs Help
year : 1971 mean : 5.5074
year : 1972 mean : 6.05164
year : 1973 mean : 5.83342
year : 1974 mean : 6.40329
year : 1975 mean : 7.00082
year : 1976 mean : 4.74754
year : 1977 mean : 5.1663
year : 1978 mean : 4.46356
year : 1979 mean : 4.55808
year : 1980 mean : 4.57896
year : 1981 mean : 4.61452
year : 1982 mean : 5.45233
year : 1983 mean : 6.06986
year : 1984 mean : 5.93169
year : 1985 mean : 3.39014
year : 1986 mean : 4.96548
year : 1987 mean : 3.55973
year : 1988 mean : 5.59918
year : 1989 mean : 7.10192
year : 1990 mean : 7.02822

```

Figure 2.9: Extract from terminal

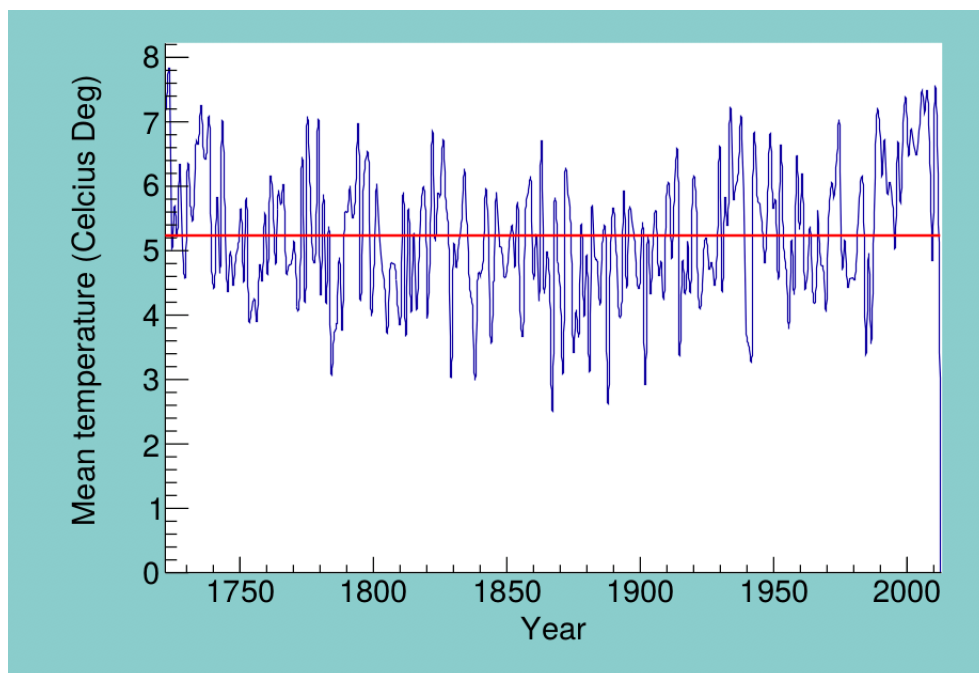


Figure 2.10: The graph shows the mean temperature of each year since 1722 in blue, and the mean of all years in red color

On the graph we can see a temperature sequence above and below the average. We notice that the temperatures are getting hotter. To improve this code we can try to extrapolate using a mathematical model to highlight global warming.

## 2.2 Temperature on a given day in Uppsala

In this section I'm going to study the temperature on a given day in Uppsala. For that we are going to use the tempdata\_uppsala.txt file. In the example we are given the choice between two functions, one takes in entry the date (in form of a number contained between 1 and 365) or a date (month and day). I choose to use the second one because I already have these informations in the data file, as shown in figure 2.11 below.

```
1  #include "tempTrender.h"
2  #include <string>
3
4
5  void project() {
6      string pathToFile = "../datasets/tempdata_uppsala.txt"; //Put the path to your data file here
7      tempTrender t(pathToFile); //Instantiate your analysis object
8
9      t.tempOnDay(8, 23); //Call some functions that you've implemented
10     //t.tempOnDay(235);
11     //t.tempPerDay();
12     //t.hotCold(2050);
13     //t.hottestTempPerYear(2050);
14     //t.tempPerYear(2050);
15 }
16
```

Figure 2.11: Function Project.cpp

In this example I'm gonna study the climate on the 23rd of August for the past 300 years. For that I wrote a code called tempTrender.cpp that takes all the informations needed in the tempdata\_uppsala.txt file and draw a plot with them. I'm now going to get through the code and explain what I did.

In the first part of the program (shown in 2.12), I'm going to declare a vector called "entries" to store every lines of the data file to have an easier access to them. Then an array for the temperatures on that day (it will be useful for the standart deviation later). And a whole bunch of variables that are going to be useful in the code.

```
32  //////////////////////////////////////
33  // JOHAN CODE PART
34  //////////////////////////////////////
35
36
37  void tempTrender::tempOnDay(int monthToCalculate, int dayToCalculate) {
38
39      ifstream file(_filePath.c_str());
40
41      //Defining all variables we need
42      vector<string> entries;
43      float temp[ARRAY_SIZE];
44
45      string line, month_str, day_str, temp_str;
46      int i, j=0, size, pos, month, day;
47      float t, mean=0, StanDev=0;
48
49      //Check if the file is opened
50      if(file.is_open())
51      {
52          //Fill up the vector with the entries
53          while(getline(file, line))
54          {
55              entries.push_back(line);
56          }
57      }
58
59      size = entries.size();
60
```

Figure 2.12: First part of code tempTrender.cpp

In this part, we are going to create a histogram and fill it with the values we get from the vector. Basically, the idea is to store the day in the variable "day" and month in the variable "month". Check if it correspond to the day we wan't to study. If it does then we store the values of the temperature on this day in the array "temp" and fill the histogram with this value. The "atoi" part is just to convert strings to integer as the vector "entries" is filled with strings.

```

64   TCanvas *c1= new TCanvas("Johan","Project : Temperature on a given day");
65
66   //creating histogram
67   TH1D* hist = new TH1D("Hist", "Temperature on a given day", 100, -20, 40);
68
69   //goes through all the entries
70   for(i=0; i<size; i++)
71   {
72       line = entries[i];
73
74       //get the month from 2nd field
75       pos = line.find(' ');
76       line.erase(0, pos + 1);
77       pos = line.find(' ');
78       month_str = line.substr(0, pos);
79       month = atoi(month_str.c_str());
80
81       //get the day from 3rd field
82       pos = line.find(' ');
83       line.erase(0, pos + 1);
84       pos = line.find(' ');
85       day_str = line.substr(0, pos);
86       day = atoi(day_str.c_str());
87
88
89       if ((month == monthToCalculate) && (day == dayToCalculate))
90       {
91           //get the corresponding temperature at the 4th field
92           pos = line.find(' ');
93           line.erase(0, pos + 1);
94           pos = line.find(' ');
95           temp_str = line.substr(0, pos);
96           t = atof(temp_str.c_str());
97           mean=mean+t;
98           temp[j]=t;
99           j=j+1;
100          hist->Fill(t);
101      }
102  }

```

Figure 2.13: Second part of code tempTrender.cpp

In the last part of the program (figure 2.14), I'm going to calculate the mean value of the temperature on that day and the standart deviation. To do so, I start with the mean value and then use a for-loop to get the standart deviation. After that, I will set up the X and Y axis and give them names, as well as display the mean value and the standart deviation on the terminal.

```

105   //Calculate the mean value on that day
106   mean=mean/j;
107
108   //Calculate the standart deviation
109   for (i=0; i<=j; i++){
110       StanDev = StanDev+(temp[i]-mean)*(temp[i]-mean)/(j+1);
111   }
112   StanDev = sqrt(StanDev);
113
114   cout << "The mean value on this day is : " << mean << endl;
115   cout << "The standart deviation is : " << StanDev << endl;
116
117   //Setting up X and Y axis
118   hist->GetYaxis()->SetTitle("Entries");
119   hist->GetXaxis()->SetTitle("Temperature in Celsius (deg)");
120
121
122   //drawing histogram
123   hist->SetFillColor(kGreen);
124   hist->Draw("SAME");
125
126
127 }
128

```

Figure 2.14: Third part of code tempTrender.cpp

As a result, the mean temperature on the 23rd of August is 14,9003 degrees celsius and the standart deviation is 2,77906 (As shown in figure 2.15).

```
The mean value on this day is : 14.9003
The standart deviation is : 2.77906
```

Figure 2.15: Mean value and standart deviation

By compiling the program, that is the result I get. The temperatures are pretty much contained between 10 and 20 degrees and we can see that the mean value is around 15 degrees.

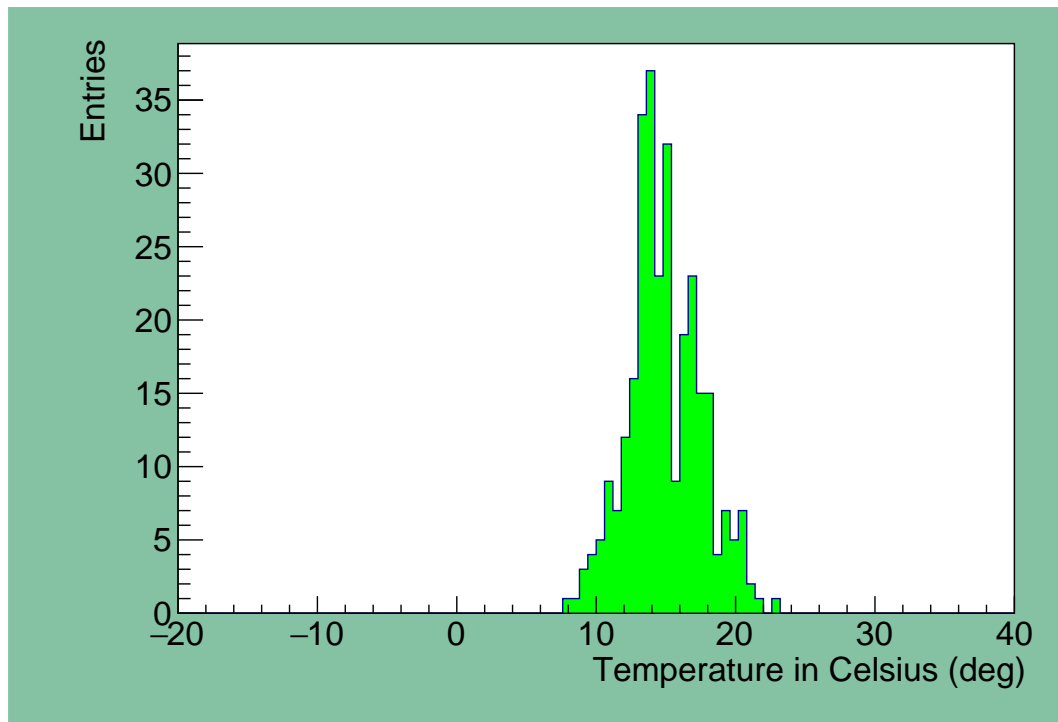


Figure 2.16: Plot of the temperature on that day

## 2.3 Hottest and coldest temperatures per year

In this section I'm going to study the coldest and hottest temperature for each years in Uppsala. For that we are going to use the `tempdata_uppsala.txt` file. In the exemple we are given the choice between two functions, one called `hottestTempPerYear` and the other called `coldestTempPerYear`. The aim is to print each hottest and coldest temperature for each year and to plot it. The two functions don't need any entries as shown in the figure 2.17 below.

```
1  #include "tempTrender.h"
2  #include <string>
3
4
5  void project() {
6      string pathToFile = "../datasets/tempdata_uppsala.txt"; //Put the path to your data file here
7      tempTrender t(pathToFile); //Instantiate your analysis object
8
9      //t.tempOnDay(8, 23); //Call some functions that you've implemented
10     //t.tempOnDay(235);
11     //t.tempPerDay();
12     //t.hotCold();
13     t.coldestTempPerYear();
14     t.hottestTempPerYear();
15     //t.tempPerYear(2050);
16 }
17
```

Figure 2.17: Function Project.cpp

In these two programs I'm gonna study the hottest and coldest temperature from 1722 to 2013. For that I wrote a code called `tempTrender.cpp` that takes all the informations needed in the `tempdata_uppsala.txt` file and draw a plot with them. I'm now going to get through the code and explain what I did. In the first part of the program (shown in 2.12). Here I'm using lot of same methods as shown above: I will *create a function that allow us to read this file and retrieve all the temperatures per year in order to calculate the mean temperature per year. So I extract the first 4 characters which correspond to the year, then the 5th field which corresponds to the temperature.* I'm going to declare a vector called `entries` where I'm going to store every line of the data file to have an easier access to them. Then an array to store the hottest temperatures on that year and a whole bunch of variables that are going to be useful in the code.

```

std::map<int, double> coldTempPerYear(std::vector<std::string> entries) {
    // temporary values
    std::string year_str, value_str;
    size_t pos;
    int year;
    double value;

    // storage
    double cold_values[MAX_YEAR]={0};

    // for each entry, extract year and value
    for(size_t y=0; y<entries.size(); y++) {
        std::string s = entries[y];
        // get year from 4 first characters
        year_str = s.substr(0, 4);
        year = atoi(year_str.c_str());

        // get value from 5th field
        for(size_t i=0; i<4; i++) {
            pos = s.find(' '); // find first delimiter
            s.substr(0, pos);
            s.erase(0, pos + 1);
        }
        pos = s.find(' ');
        value_str = s.substr(0, pos);
        value = atof(value_str.c_str());

        // add value to arrays
        if (value < cold_values[year] ) {
            cold_values[year] = value;
        }
    }
}

```

Figure 2.18: Coldest temperature per year

```

std::map<int, double> hotTempPerYear(std::vector<std::string> entries) {
    // temporary values
    std::string year_str, value_str;
    size_t pos;
    int year;
    double value;

    // storage
    double hot_values[MAX_YEAR]={0};

    // for each entry, extract year and value
    for(size_t y=0; y<entries.size(); y++) {
        std::string s = entries[y];
        // get year from 4 first characters
        year_str = s.substr(0, 4);
        year = atoi(year_str.c_str());

        // get value from 5th field
        for(size_t i=0; i<4; i++) {
            pos = s.find(' '); // find first delimiter
            s.substr(0, pos);
            s.erase(0, pos + 1);
        }
        pos = s.find(' ');
        value_str = s.substr(0, pos);
        value = atof(value_str.c_str());

        // add value to arrays
        if (value > hot_values[year] ) {
            hot_values[year] = value;
        }
    }
}

```

Figure 2.19: Hottest temperature per year

So, I used “atoi” and “atof” command to convert strings to integer and to convert integer to string. Because when you are working with vectors you have to use an integer as the vector “entries”. You can also notice that I used a very simple if-loop inside the for-loop to get the extreme temperatures. In orders to replace the value inside the array by the new one of this year only if it’s a temperature hotter for the hottestTempPerYear function (colder for the coldestTempPerYear function). Finally, I will have an array that will store the hottest/coldest temperatures corresponding to specified years. In this part, we are going to create a histogram and fill it with the values we get from the vector created. Basically, the idea is to store the coldest in the variable “cold” and hottest in the variable “hot”. The “atoi” part is also to convert strings to integer as the vector “entries” is filled with strings. We will see later what the “std::cear « year : “ « I « “coldest temp : “ « cold « std::endl;” is printing. (Figure ...)

```

// return variable
std::map<int, double> ret;
double cold;

// for each year, compute coldest value
for(size_t i=0; i<MAX_YEAR; i++) {
    cold = cold_values[i];
    std::cerr << "year : " << i << " coldest temp : " << cold << std::endl;
    ret.insert(std::make_pair<int, double>((int)i, (double)cold));
}
return ret;
}

void tempTrender::coldestTempPerYear() {

    // open input file
    std::string line;
    ifstream inputfile(_filePath.c_str());

    // create array to store values
    std::vector<std::string> entries;

    // read values and stor it into array
    if(inputfile.is_open()) {
        while(getline(inputfile, line)) {
            entries.push_back(line);
        }
    }

    std::map<int, double> cold = coldTempPerYear(entries);

    // create canvas for graph
    TCanvas *c1 = new TCanvas("Jasmain", "Project : Coldest temperature over each year")

    // create new histogram object
    TH1D* hist = new TH1D("hist", "Coldest Temp Per Year", cold.size(), 1722, 2013);

    for( std::map<int, double>::iterator it = cold.begin();
        it != cold.end();
        it++ ) {
        // fill hist with date and value from mean temp per year map
        //std::cerr << "value : " << it->second << std::endl;
        hist->Fill(it->first, it->second);
    }
}

```

Figure 2.20: Main function

Then, I juste have to define and describe my histogram, with my title and the variables which constitute the x and y axis numbers. So, we can draw the histogram for both functions.

year : 1729 hottest temp : 19.1	year : 1858 coldest temp : -16
year : 1730 hottest temp : 24.8	year : 1859 coldest temp : -25.9
year : 1731 hottest temp : 20.8	year : 1860 coldest temp : -20.4
year : 1732 hottest temp : 20.5	year : 1861 coldest temp : -23.8
year : 1733 hottest temp : 22.7	year : 1862 coldest temp : -20.4
year : 1734 hottest temp : 22.1	year : 1863 coldest temp : -11.3
year : 1735 hottest temp : 22.5	year : 1864 coldest temp : -13.4
year : 1736 hottest temp : 23.2	year : 1865 coldest temp : -24
year : 1737 hottest temp : 21	year : 1866 coldest temp : -15
year : 1738 hottest temp : 20.6	year : 1867 coldest temp : -17.1
year : 1739 hottest temp : 21.5	year : 1868 coldest temp : -21.7
year : 1740 hottest temp : 20.6	year : 1869 coldest temp : -14.4
year : 1741 hottest temp : 21.9	year : 1870 coldest temp : -20.1
year : 1742 hottest temp : 22.2	year : 1871 coldest temp : -23.6
year : 1743 hottest temp : 22.4	year : 1872 coldest temp : -12.6
year : 1744 hottest temp : 21.2	year : 1873 coldest temp : -12.7
year : 1745 hottest temp : 21.6	year : 1874 coldest temp : -18.6
year : 1746 hottest temp : 22	year : 1875 coldest temp : -31.1
year : 1747 hottest temp : 23.6	year : 1876 coldest temp : -22.1
year : 1748 hottest temp : 25.1	year : 1877 coldest temp : -21.2
year : 1749 hottest temp : 21.8	year : 1878 coldest temp : -15.9
year : 1750 hottest temp : 22.7	year : 1879 coldest temp : -21.1
year : 1751 hottest temp : 21.8	year : 1880 coldest temp : -15.8
year : 1752 hottest temp : 26.9	year : 1881 coldest temp : -24.3

Figure 2.21: Output value of coldest and hottest temperature for each year

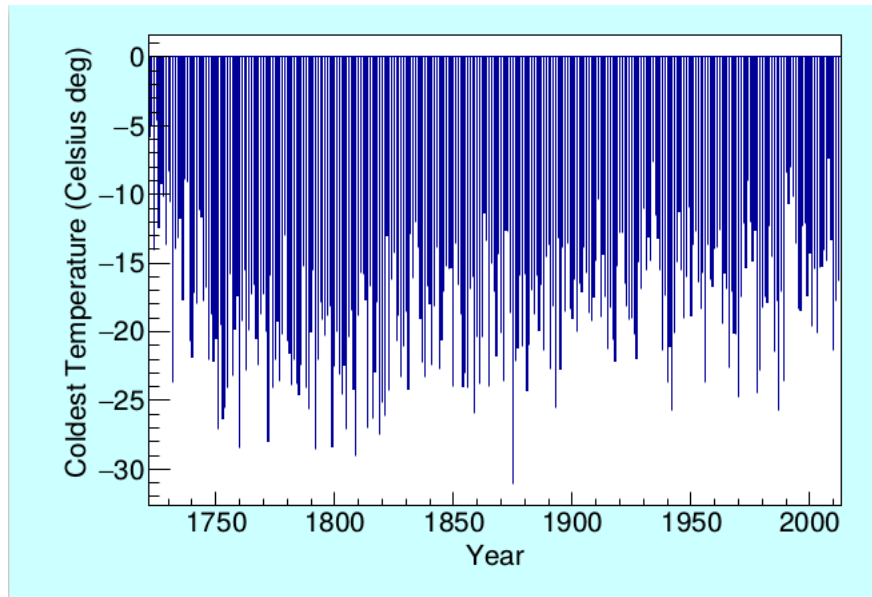


Figure 2.22: Coldest temperature

Here, above and bellow we have the different results of our program. First, (Figure 2.22) we have all the values of the hottest temperatures/coldest temperatures for each year between 1722 and 2013.

Besides, we have the 2 different histogram that show us how the hottest and coldest temperature has evolved during the last centuries. We can analyse that it's very hard to occur the hottest and coldest temperature even if we know that we have a global-warming. In one century we will probably get a much better view of the global warming on this histogram.



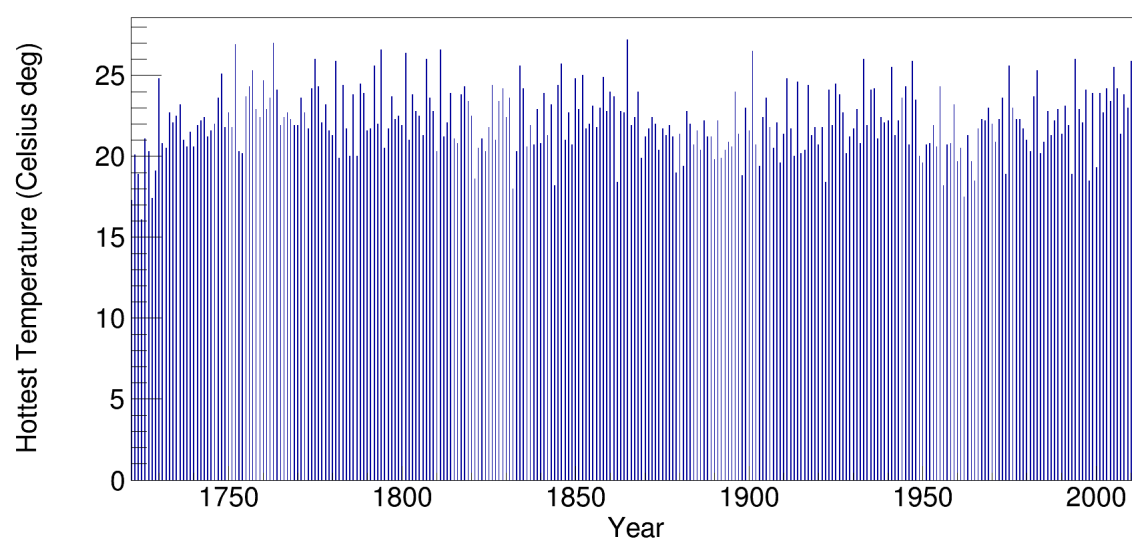
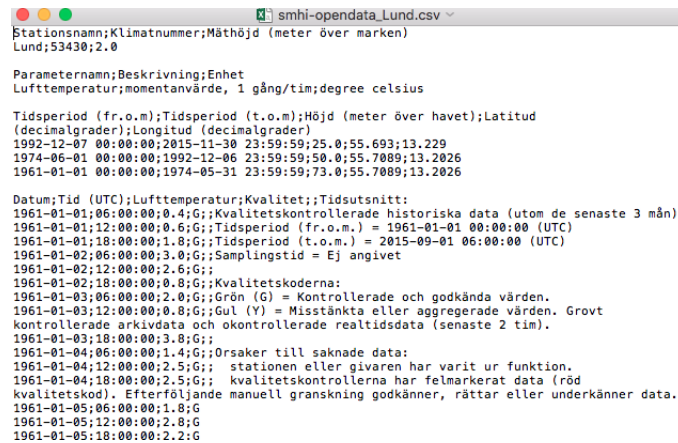


Figure 2.23: Hottest temperature

## Chapter 3

# Temperatures per days in Lund, Umea and Falsterbo

For this part of the project, I'm gonna use the files associated with Lund, Umea and Falsterbo, figure 3.1 shows a screenshot of the file associated with Lund.



```
Stationsnamn;Klimatnummer;Måthöjd (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)
1992-12-07 00:00:00;2015-11-30 23:59:59;25.0;55.693;13.229
1974-06-01 00:00:00;1992-12-06 23:59:59;50.0;55.7089;13.2026
1961-01-01 00:00:00;1974-05-31 23:59:59;73.0;55.7089;13.2026

Datum;Tid (UTC);Lufttemperatur;Kvalitet;;Tidsutsnitt:
1961-01-01;06:00:00;0.4;G;;Kvalitetskontrollerade historiska data (utom de senaste 3 mån)
1961-01-01;12:00:00;0.6;G;;Tidsperiod (fr.o.m.) = 1961-01-01 00:00:00 (UTC)
1961-01-01;18:00:00;1.8;G;;Tidsperiod (t.o.m.) = 2015-09-01 06:00:00 (UTC)
1961-01-02;06:00:00;3.0;G;;Samplingstid = Ej angivet
1961-01-02;12:00:00;2.6;G;;
1961-01-02;18:00:00;0.8;G;;Kvalitetskoderna:
1961-01-03;06:00:00;2.0;G;;Grön (G) = Kontrollerade och godkända värden.
1961-01-03;12:00:00;0.8;G;;Gul (Y) = Misstänkta eller aggregerade värden. Grovt
kontrollerade arkivdata och okontrollerade realtidsdata (senaste 2 tim).
1961-01-03;18:00:00;3.8;G;;
1961-01-04;06:00:00;1.4;G;;Orsaker till saknade data:
1961-01-04;12:00:00;2.5;G;; stationen eller givaren har varit ur funktion.
1961-01-04;18:00:00;2.5;G;; kvalitetskontrollerna har felmarkerat data (röd
kvalitetskod). Efterföljande manuell granskning godkänner, rättar eller underkänner data.
1961-01-05;06:00:00;1.8;G
1961-01-05;12:00:00;2.8;G
1961-01-05;18:00:00;2.2;G
```

Figure 3.1: The raw data stored in a CSV file.

To extract only the measurements of the file, that is, all that follows after the line starting with "Datum; ..." and only the first four columns (using the field separator ";" for the definition of a column), a bash script tempdata.sh was written and executed. This script would extract the relevant data and save it to a new file data\_for\_town.txt. The script went through a couple of drafts, yielding different outputs as shown in figure 3.2.

```

data_for_Lund.txt
Datum Tid (UTC) Lufttemperatur Kvalitet
1961-01-01 06:00:00 0.4 G
1961-01-01 12:00:00 0.6 G
1961-01-01 18:00:00 1.8 G
1961-01-02 06:00:00 3.0 G
1961-01-02 12:00:00 2.6 G
1961-01-02 18:00:00 0.8 G
1961-01-03 06:00:00 2.0 G
1961-01-03 12:00:00 0.8 G
1961-01-03 18:00:00 3.8 G
1961-01-04 06:00:00 1.4 G

```

(a)

```

data_for_Lund2.txt
19610101 060000 0.4 G 1
19610101 120000 0.6 G 1
19610101 180000 1.8 G 1
19610102 060000 3.0 G 2
19610102 120000 2.6 G 2
19610102 180000 0.8 G 2
19610103 060000 2.0 G 3
19610103 120000 0.8 G 3
19610103 180000 3.8 G 3
19610104 060000 1.4 G 4
19610104 120000 2.5 G 4

```

(b)

```

data_for_Lund3.txt
1961 1 1 6 0.4 G 1
1961 1 1 12 0.6 G 1
1961 1 1 18 1.8 G 1
1961 1 2 6 3.0 G 2
1961 1 2 12 2.6 G 2
1961 1 2 18 0.8 G 2
1961 1 3 6 2.0 G 3
1961 1 3 12 0.8 G 3
1961 1 3 18 3.8 G 3
1961 1 4 6 1.4 G 4
1961 1 4 12 2.5 G 4
1961 1 4 18 2.5 G 4

```

(c)

Figure 3.2: Different outputs of tempdata.sh; from earlier drafts, ?? and ??, and final draft, ??.

The last column in ?? and ?? specifies the day number of the year, i.e. a number between 1 and 365/366. This was achieved by creating a UNIX timestamp in tempdata.sh, which, by the way, used the single command GNU awk to format the data (see figure 3.3).

```

#!/bin/sh
#author: Philip Siemund
#Extracts only the temperature data from SMHI datasets and saves the output in a new file data_for_${DATASET}.txt
#Requires GNU awk or mawk. Check your paths!

DATASET=$@
OUTPUTFILENAME=data_for_${DATASET}

if [[ -z "$DATASET" ]]; then
    echo "Please insert valid argument."
    exit 1
fi

awk -F';' '
{
    x=1
    gsub(/-/, "", $1)
    gsub(/:/, "", $2)
    gsub(/^\^/, "", $2)
    tspec = sprintf("%4d %2d %2d 00 00 00", substr($1,1,4), substr($1,6,2), substr($1,9,2))
    t = mktime(tspec)
    gsub(" ", "", $1)
    print $1, $2, $3, $4, 0 + strftime("%j", t)
}
' /Datum/ {x=1} ../datasets/smhi-opendata_${DATASET}.csv >> ../datasets/${OUTPUTFILENAME}.txt

```

Figure 3.3: tempdata.sh.

The purpose behind tempdata.sh was to make the data more accessible. Including the header fstream in a given C file, one could simply stream the different datatypes in the file data\_for\_town.txt onto corresponding variables in a while-loop. The data of interest could be specified by including conditional statements in the while-loop.

The file analyze\_data.C exemplifies this (see figure 3.4). If one compiles the file in the data analysis tool ROOT and calls the function temPerDay(), it produces a histogram showing the temperature per day in a town at a given hour a given year. Some such histograms are shown in figure 3.5.

```
//author: Philip Siemund
//Draws a 1D histogram showing temp. per day at a specified time during a year. The raw data
//is extracted from the datasets using a bash script tempdata.sh. The bash script is executed once whenever
//the argument const char* town is specified for the first time for a given town. Check your paths!

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

using namespace std;

#include </Users/philipsiemund/root_v6.18.04/include/TH1.h>
#include </Users/philipsiemund/root_v6.18.04/include/TH2.h>
#include </Users/philipsiemund/root_v6.18.04/include/TCanvas.h>
#include </Users/philipsiemund/root_v6.18.04/include/TLatex.h>
#include </Users/philipsiemund/root_v6.18.04/include/TSystem.h>

void temPerDay(Int_t year, Int_t hour, const char* town){

    string fileName = Form("/Users/philipsiemund/MNXB01-Project-2019/datasets/data_for_%s.txt", town);

    ifstream tempo(fileName);

    if(!tempo) {
        cout << "Error: could not read from file ..." << endl;
        cout << "Running tempdata.sh ..." << endl;
        gSystem->Exec(Form("./tempdata.sh %s", town));
    }

    tempo.close();
    ifstream file(fileName);

    TH1D* hist = new TH1D("Stats", Form("Temp. per day in %s at %d:00 UTC in %d; Day; Temperature [#circC]",town,hour,year),
        366, 0, 366);
```

(a)

```
Int_t Year;
Int_t month;
Int_t day;
Int_t time;
Double_t temp;
string quality;
Int_t dayno;

while (file >> Year >> month >> day >> time >> temp >> quality >> dayno){

    if (Year == year && time == hour) {

        hist->SetBinContent(dayno,temp);
    }

    else {

        continue;
    }
}

file.close();

TCanvas* can = new TCanvas("can", "hist canvas", 900, 600);
hist->SetLineColor(1);
hist->Draw();

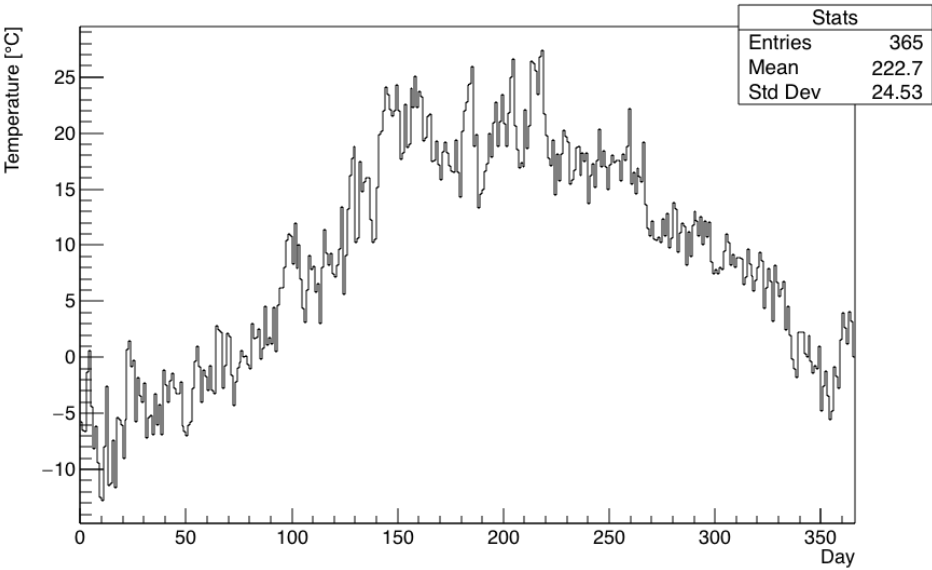
can->SaveAs("hist.png");

}
```

(b)

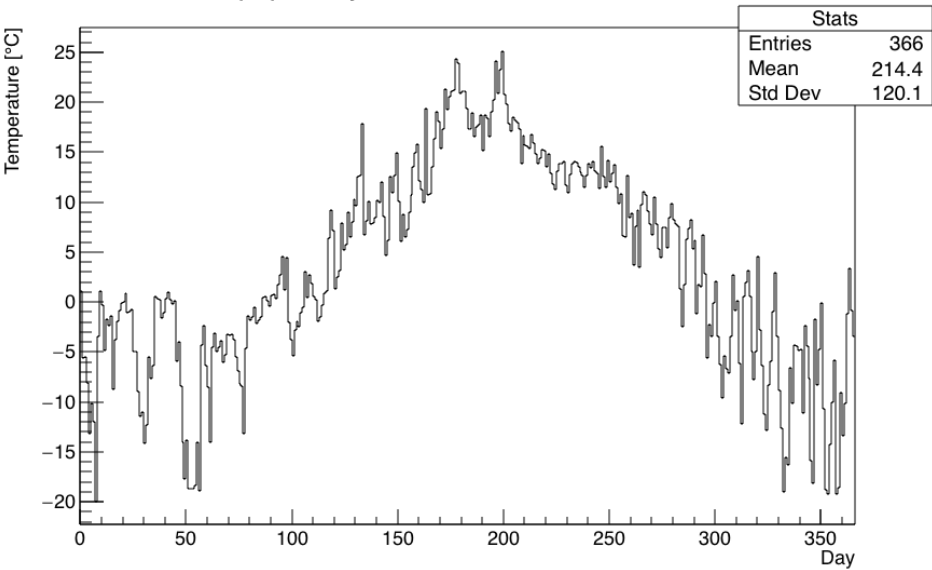
Figure 3.4: analyze\_data.C.

Temp. per day in Lund at 12:00 UTC in 1963



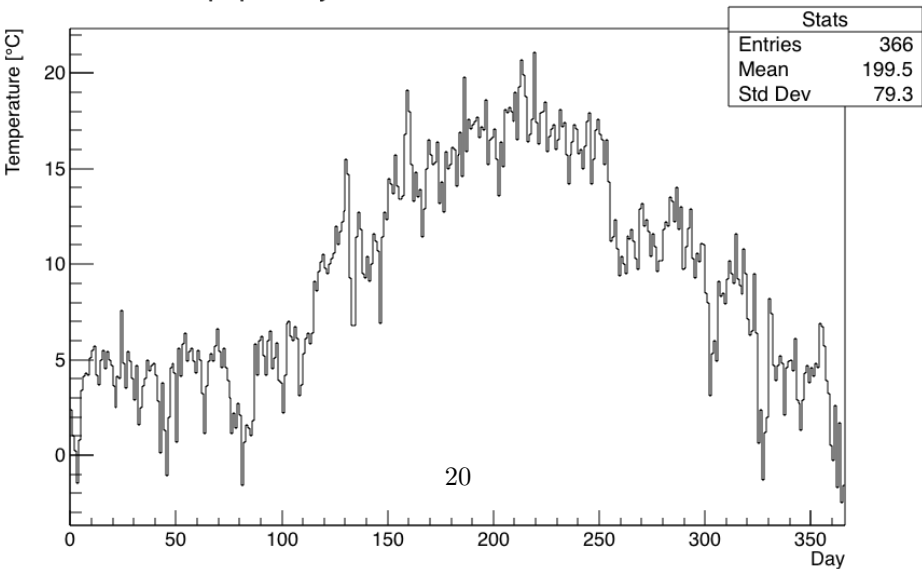
(a)

Temp. per day in Umea at 18:00 UTC in 1988



(b)

Temp. per day in Falsterbo at 21:00 UTC in 2008



(c)