**„Programming”**

**C3**

*Made by: Cristea Andrei*

*Neptun code: W61RAB*

*E-mail:* [*cristea.andrei997@gmail.com*](mailto:cristea.andrei997@gmail.com)



*Course code: IP-18fPROGEG*

*Teacher’s name: Pluhár Zsuzsa*

***2020. June 05***

Content

[User documentation](#bookmark=id.hjtwmlch0joc) 3

[Task](#bookmark=id.xu6znovff5jd) 3

[Runtime environment](#bookmark=id.2d2xint8v4rk) 3

[Usage](#bookmark=id.jtv2kdlep6sx) 3

[Starting the program](#bookmark=id.3yafj8mnoiod) 3

[Program input](#bookmark=id.h6vhqbsiv70p) 3

[Program output](#bookmark=id.f4lu00d14kxw) 3

[Sample input and output](#bookmark=id.ut5u5axbhfq) 4

[Possible errors](#bookmark=id.fj7r9a6pu6k9) 4

[Developer documentation](#bookmark=id.5cbex7vzuz08) [6](#bookmark=id.tyjcwt)

[Task](#bookmark=id.wy2q2hi3lfo6) [6](#bookmark=id.tyjcwt)

[Specification](#bookmark=id.ti47n9whg9s7) [6](#bookmark=id.tyjcwt)

[Developer environment](#bookmark=id.jkc5tw8j4cxp) [7](#bookmark=id.3dy6vkm)

[Source code](#bookmark=id.ldsts5p6oj1v) [7](#bookmark=id.3dy6vkm)

[Solution](#bookmark=id.2x62whyilhac) [7](#bookmark=id.3dy6vkm)

[Program parameters](#bookmark=id.7sq8mg8ae5tk) [7](#bookmark=id.3dy6vkm)

[The structure of the program](#bookmark=id.bcz1d7hmxblx) 8

[Structure of functions](#bookmark=id.cy6dxv2m58uw) 9

[The algorithm of the program](#bookmark=id.4vac7suxf09e) 10

[The code](#bookmark=id.5qc9yaiz121k) 13

[Testing](#bookmark=id.x8xklghb4b9p) 22

[Valid test cases](#bookmark=id.b960pp3ozwvv) 22

[Invalid test cases](#bookmark=id.17dp8vu) 24

[Further development options](#bookmark=id.eum4kkdwz3l) 25

**User documentation**

**Task**

### Task Name : Continuously the warmest settlement

We have the weather forecast for the next N days for M settlements.

Write a program that gives the warmest settlement. The warmest settlement is the settlement

which has the most count of days when its temperature is higher than temperatures

of all the other settlements.

**R****untime environment**

A Macbook Pro 2014 that is run on Mac OS 64 system. For running program you can use any other operating system. No mouse needed.

**U****sage**

**St****arting the program**

The program can be found in the archived file by the name: warmest\program\warmest.exe

If you are using Linux/ Mac OS: warmest\program\warmest.out

You can start the program by clicking the warmest.exe file.

**Pro****gram input**

The program reads the input data from the keyboard in the following order:

|  |  |  |
| --- | --- | --- |
| **#** | **Data** | **Explanation** |
| **1.** | *N* | Number of days weather was forecasted (1 <= N <= 1000). |
| **2.** | *M* | Count of settlements (0 <= M <= 1000). |
| **3.** | *L* | A temperature limit (20 <= L <= 50). |
| **...** | *…* |  |
| **(N+1)\*(M+1)** | *temperatureN,M* | The Nth settlements the Mth days temperature (0 <= temperature <=50). |

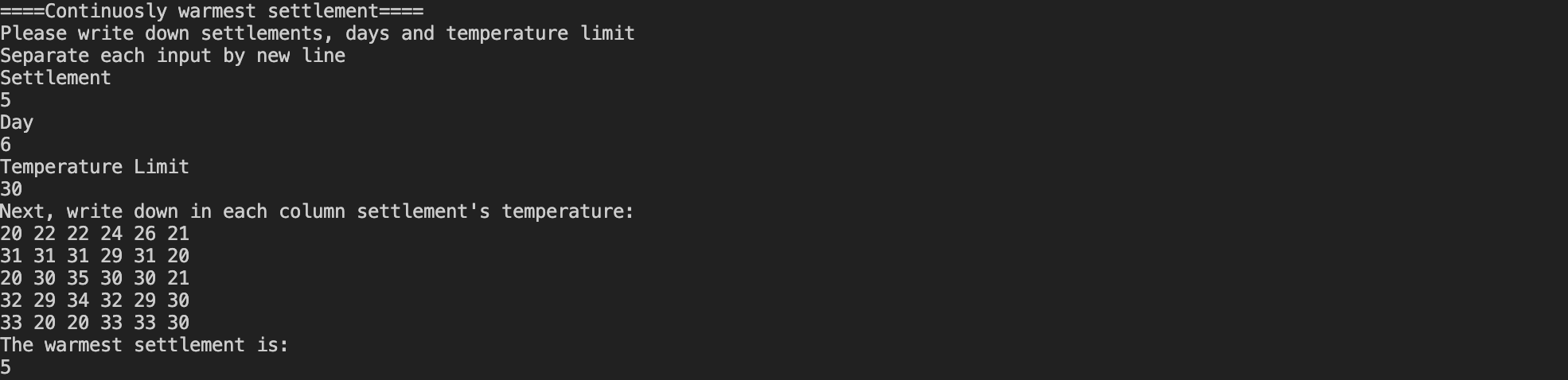


**Pro****gram output**

The program writes out the index of settlement which is the hottest in human readable format(<=1) . The hottest settlement is the one which has the most count of days when the temperature is hotter then in selected day in other settlements.

If there is multiple solutions output should be the one with the smallest index

**Sam****ple input and output**



**Po****ssible errors**

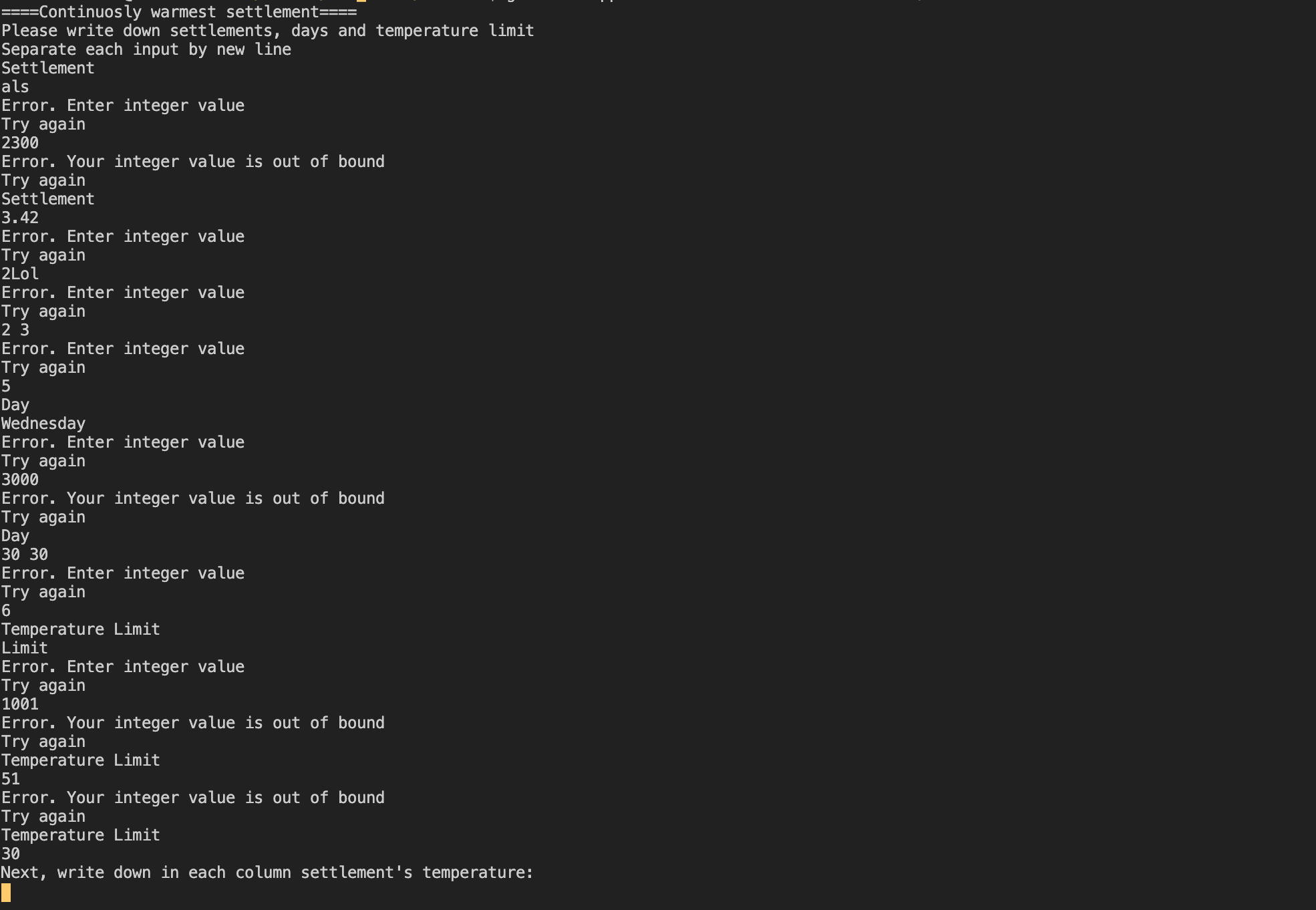
The input should be given according to the sample. If the number of measurements is not a whole number, or it is not in the range 2..10000, it will cause a problem. If one of the measurements is not a number, or it is not in the range 0..9000, it also will cause a problem. In the case of an error, the program displays an error message, or asks for the repetition of the input. There are two types of streams in input for the program: integer values and array values. In order to ease debugging process and to paste big data inside array of temperatures second input validation function will work with getline from STL. Samples of invalid data validation is below.

***Sampl******e of running in the case of invalid data for settlement, day***

***and temperature limit:***

Inputs which can’t be validated for N, M and L:

* Integer is out of range for e.g. range can be: [1..1000]
* Any other types except integer
* More than one integer in a line

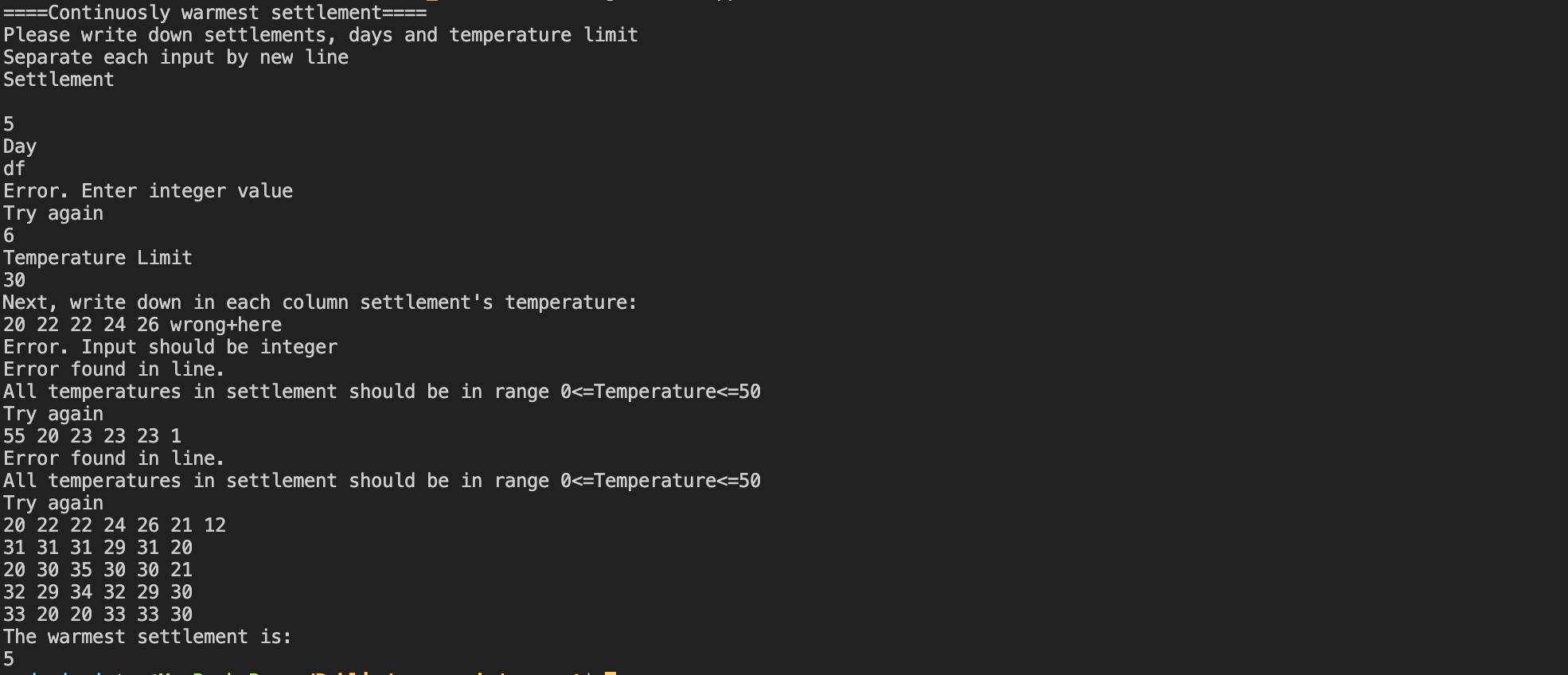


***Sampl******e of running in the case of invalid data for temperature sequence.***

***Here validator checks entire line for the error.***

Inputs which can’t be validated for weather1..Ν,1..M:

* Integers are out of range: [0..50]
* Any other types except integers
* Ignores Values which are out of array size



**Devel****oper documentation**

**Ta****sk**

We have the weather forecast for the next N days for M settlements.

Write a program that gives the warmest settlement. The warmest settlement is the settlement

which has the most count of days when its temperature is higher than temperatures

of all the other settlements.

Remark:

If there is multiple solutions output should be the one with the smallest index

**Spec****ification**

**Input**: N, M, L ∈N, weather1..Ν,1..M ∈NN\*M

**Output**: biggest ∈N

**Precondition**: N = [1..1000] ∧ M = [1..1000] ∧ L=[20..50]

**Postcondition:** ∀j(1<=j<=M) and ∀i(1<=i<=N): transposej,i = weatheri,j ⋁

1<= MaxInd <= N and ∀j(1<=i<=M) and ∀i(1<=j<=N): transposei, maxInd > transposei, j

and max = transposei, maxInd and index = j and has\_no\_duplicates(transposei, j,transposei,j )

and max > L

∀j(1<=i<=M) and ∀i(1<=j<=N): not has\_no\_duplicates(transposei, j,transposei,j ) : duplicates.insert()

⋁ ind\_dup :=ind\_duplicate(transposei) and

M M

outindex= ⅀1 and outtake\_dup\_index(transposei, transposei,ind\_dup)= ⅀1

i=1 i=1

max > L or all\_less\_than\_L duplicates.count(i) and transposei,ind\_dup

⋁ biggest := take\_biggest(out)

**Definitions**:

**has\_no\_duplicates :** Set( transposei) = transposei

**give\_biggest**: ∀i(1<=i<=N): 1<= biggest<= N and ∀j(1<=i<=N) :out, i > cntj

and cnt=out maxInd

**take\_dup\_index**: ∃i(1<=i<=N) :v[i] = d

**find\_duplicate**: ∃j(1<=j<=N) and ∃i(1<=i<=N): v[i] = v[j]

**all\_less\_than\_L**: ∀i(1<=i<=M) and ∀j(1<=j<=N):∀ v[i][j] <= L

**Devel****oper environment**

**Original Developed Environment**:

Macbook Pro, and operating system capable of running .out files(Mac OS Mojave). Apple clang version 11.0.0 (clang-1100.0.33.8) compiler, Xcode(v 11.0) developer tool(mainly used for debugging). Additionally used for input clarification Visual Studio Code(v 1.45.1).

**Developer’s Recommended Environment**

Any operating system with pre-installed compiler g++.

e.g:

IBM PC, an operating system capable of running exe files (eg. Windows 7). mingw32-g++.exe c++ compiler (v4.7), Code::Blocks (v13.12) developer tool.

**Sour****ce code**

All the sources can be found in the *A1B2C3* folder (after extraction). The folder structure used for development:

|  |  |
| --- | --- |
| **File** | **Explanation** |
| *warmest\program\warmest.exe*  *warmest\program\warmest.out* | Executable code for Windows  Executable code for Linux & Mac Os |
| *warmest\object\main.o* | Semi-compiled code |
| *warmest\main.cpp* | C++ source code |
| *warmest\tests\test1.txt* | input test file1 |
| *warmest\tests\test11.txt* | input test file11 |
| *warmest\tests\test2.txt* | input test file2 |
| *warmest\tests\test3.txt* | input test file3 |
| *warmest\tests\test4.txt* | input test file4 |
| *warmest\tests\test5.txt* | input test file5 |
| *warmest\tests\test6.txt* | input test file6 |
| *warmest\docs\warmest\_settlement.docx* | documentation (this file) |
| *warmest\docs\warmest\_settlement.pdf* | documentation (this file) (pdf format) |

**Solu****tion**

**Program parameters**

***Con******tants***

min\_limit : **Integer**(1)

max\_limit : **Integer**(1000)

Types

weather = **Array**(1..N:**Integer,** 1..M: **Integer**)

transpose = **Array**(1..M:**Integer,** 1..N: **Integer**)

out = **Array**(1..N: **Integer**)

duplicates = **Set**(1..cnt: **Integer**)

T = **Template**

Local(looped types)

temp = **Array(1..split\_to\_int.size()**: **Integer)**

***Variables***

N : **Integer**

M : **Integer**

L : **Integer**

biggest : **Integer**

Local(looped variables)

all\_less : **Boolean**

repeat : **Boolean**

str : **String**

max : **Integer**

index : **Integer**

dup\_ind : **Integer**

**The stru****cture of the program**

The modules used by the program, and their locations:

main.cpp – the program, in the source folder

iostream – keyboard and console management, part of the C++ system

vector – special type of container with useful methods , part of the C++ system

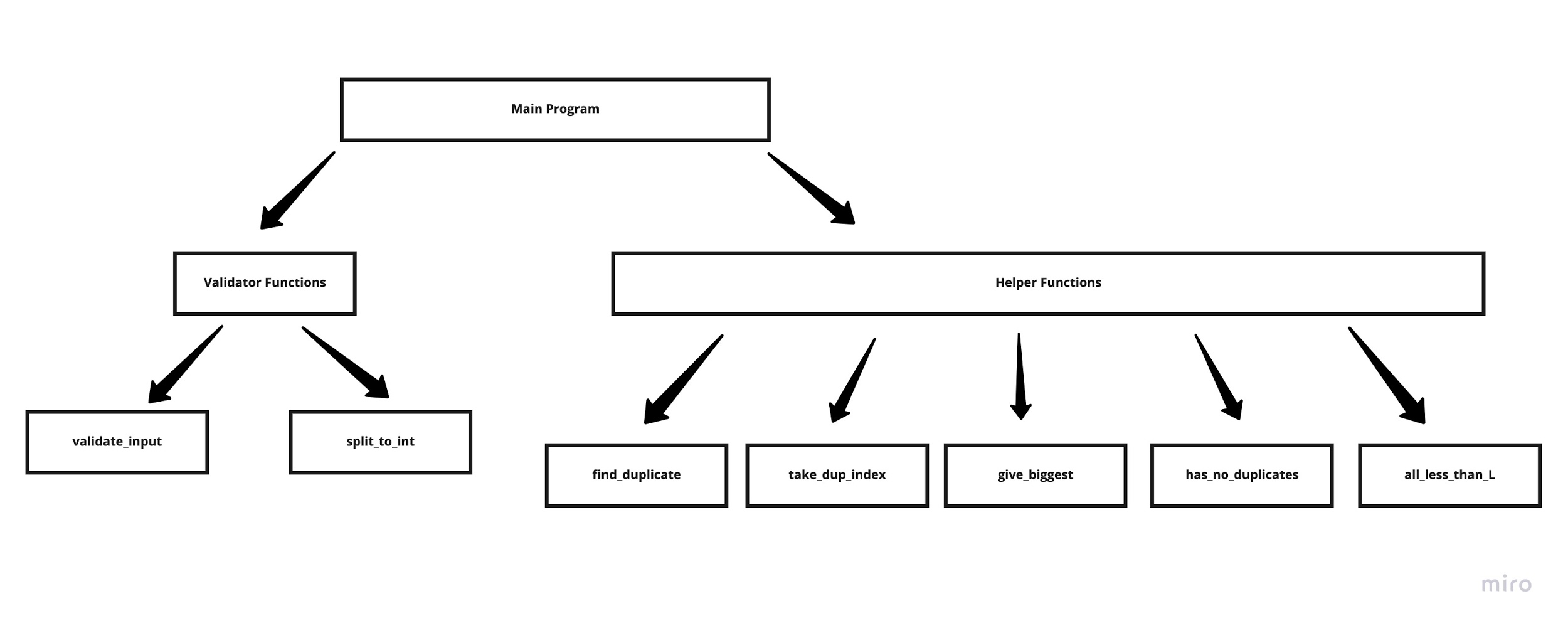
set – special type of container in C++ which has same concepts of set as in maths, part of the C++ system

limits - defines numeric\_limits, part of the C++ system

string - header file with functions classes and variable for organizing work with strings, part of the C++ system.

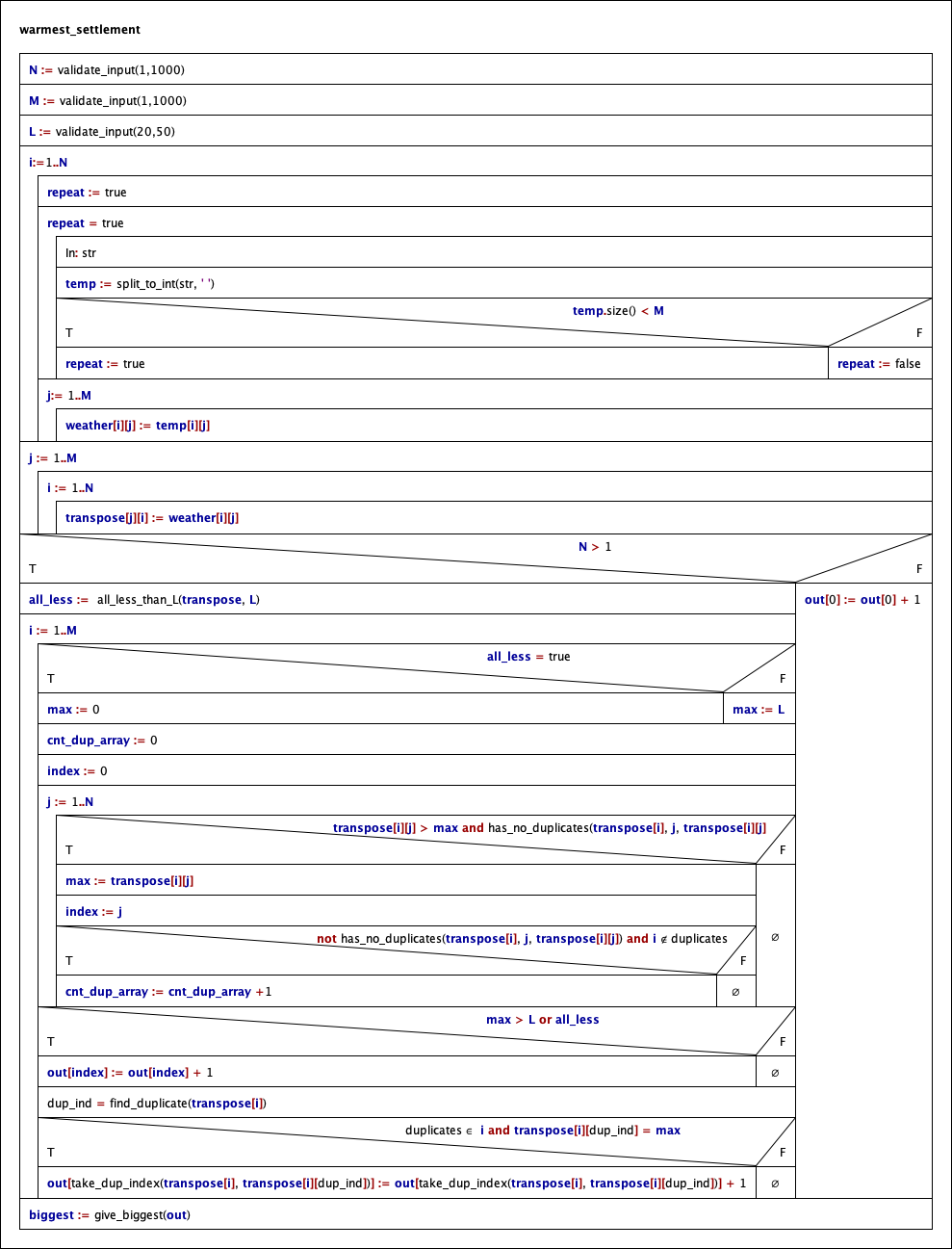
sstream- header file with functions classes and variable for organizing work with strings in streams, part of the C++ system.

**Struct****ure of function**



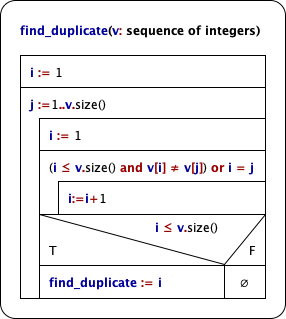
**The al****gorithm of the program**

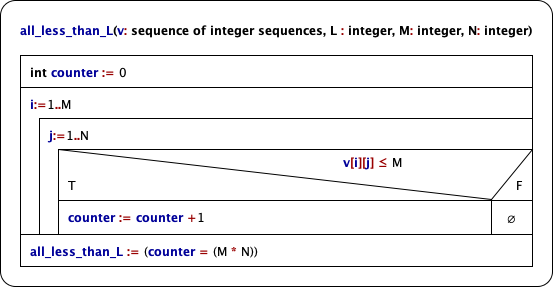
Main program:

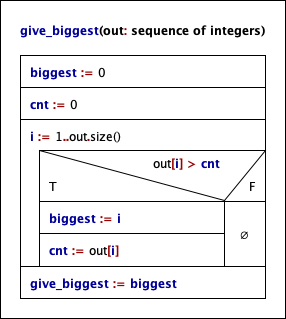


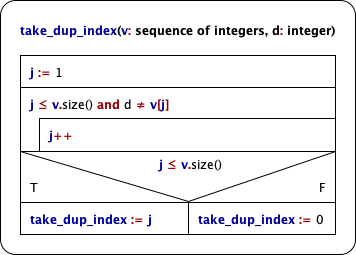
Subprograms:

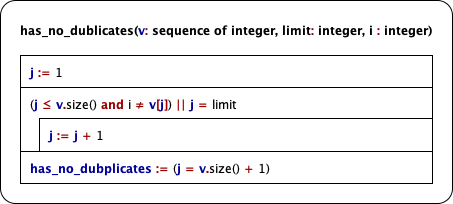
**Helper Functions:**



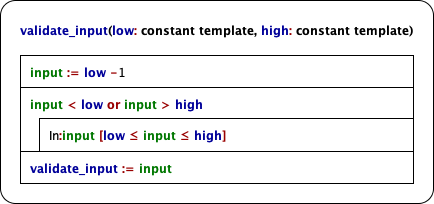


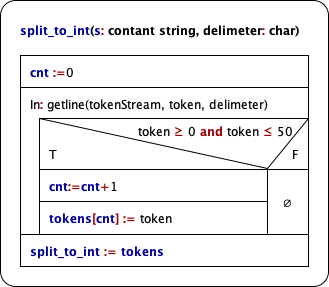






**Input Valida****tion Functions:**





**The** **code**

The content of the main.cpp file:

/\*

Created by: Andrei Cristea

Neptun: W61RAB

E-mail: cristea.andrei997@gmail.com

Task: „C3" - Continuously the warmest settlement

\*/

#include <iostream>

#include <vector>

#include <set>

#include <sstream>

#include <limits>

#include <string>

**using****namespace** std;

**template** <**class****T**>

T validate\_input(**const** T low, **const** T high, **const** string &message, **const** string &error, **const** string &mismatch)

{

T input = low - **1**;

**while** (input < low || input > high)

{

cout << message;

**while** ((cin >> input).fail() || cin.peek() != '\n')

{

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cerr << error;

}

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

**if** (input < low || input > high)

{

cerr << mismatch;

}

}

**return** input;

}

vector<**int**> split\_to\_int(**const** string &s, **char** delimeter)

{

vector<**int**> tokens;

string token;

istringstream **tokenStream**(s);

**while** (getline(tokenStream, token, delimeter))

{

try

{

**if** (stoi(token) >= **0** && stoi(token) <= **50**)

{

tokens.push\_back(stoi(token));

}

}

**catch** (exception &err)

{

cerr << "Error. Input should be integer**\n**";

**break**;

}

}

**return** tokens;

};

**bool****all\_less\_than\_L**(vector<vector<**int**> > v, **int** M){

**int** counter = **0**;

**for** (**int** i = **0**; i < v.size(); i++)

{

**for**(**auto** vi : v[i]){

**if**(vi <= M){

counter++;

}

}

}

**return** (counter == (v.size() \* v[**0**].size()));

}

**int****find\_duplicate**(vector<**int**> v)

{

**int** i = **0**;

**for** (**int** j = **0**; j < v.size(); j++)

{

i = **0**;

**while** ((i < v.size() && v[i] != v[j]) || i == j)

{

i++;

}

**if** (i < v.size())

{

**break**;

}

}

**return** i;

};

**int****take\_dup\_index**(vector<**int**> v, **int** d)

{

**int** j = **0**;

**while** (j < v.size() && d != v[j])

{

j++;

}

**if** (j < v.size())

{

**return** j;

}

**return****0**;

};

**int****give\_biggest**(vector<**int**> out)

{

**int** biggest = **0**;

**int** cnt = **0**;

**for** (**int** i = **0**; i < out.size(); i++)

{

**if** (out[i] > cnt)

{

biggest = i;

cnt = out[i];

}

}

**return** biggest;

};

**bool****has\_no\_duplicates**(vector<**int**> v, **int** limit, **int** i)

{

**int** j = **0**;

**while** ((j < v.size() && i != v[j]) || j == limit)

{

j++;

}

**return** (j == v.size());

};

**int****main**()

{

**int** N = **0**, M = **0**, L = **0**;

cout << "====Continuosly warmest settlement====**\n**";

cout << "Please write down settlements, days and temperature limit**\n**";

cout << "Separate each input by new line**\n**";

**const****int** min\_limit = **1**;

**const****int** max\_limit = **1000**;

**const** string type\_error = "Error. Enter integer value**\n**Try again**\n**";

**const** string limit\_error = "Error. Your integer value is out of bound**\n**Try again**\n**";

N = validate\_input(min\_limit, max\_limit, "Settlement**\n**", type\_error, limit\_error);

M = validate\_input(min\_limit, max\_limit, "Day**\n**", type\_error, limit\_error);

L = validate\_input(**20**, **50**, "Temperature Limit**\n**", type\_error, limit\_error);

cout << "Next, write down in each column settlement's temperature:**\n**";

vector<vector<**int**> > weather(N, vector<**int**>(M, **0**));

**for** (**int** i = **0**; i < N; i++)

{

**bool** repeat = true;

vector<**int**> temp;

**while** (repeat)

{

string str;

getline(cin, str, '\n');

temp = split\_to\_int(str, ' ');

**if** (temp.size() < M)

{

repeat = true;

cerr << "Error found in line.**\n**All temperatures in settlement should be in range 0<=Temperature<=50**\n**Try again**\n**";

}

**else**

repeat = false;

}

**for** (**int** j = **0**; j < M; j++)

{

weather[i][j] = temp[j];

}

}

// First, transpose matrix in order to have simplify task

vector<vector<**int**> > transpose(M);

**for** (**int** j = **0**; j < M; j++)

{

**for** (**int** i = **0**; i < N; i++)

{

transpose[j].push\_back(weather[i][j]);

}

}

// Initialize helper data collections

vector<**int**> out(N);

set<**int**> duplicates;

// Main computation part

**if** (N > **1**)

{

**bool** all\_less = all\_less\_than\_L(weather, L);

// looping over transpose matrix in order to find in each column maximum temperatures

**for** (**int** i = **0**; i < transpose.size(); i++)

{

**int** max = **0**;

**if** (all\_less)

{

max = **0**;

}**else**{

max = L;

}

**int** index = **0**;

**for** (**int** j = **0**; j < transpose[i].size(); j++)

{

// Take maximum temperature, which is unique (biggest temp can not have duplicates)

**if** (transpose[i][j] > max && has\_no\_duplicates(transpose[i], j, transpose[i][j]))

{

max = transpose[i][j];

index = j;

}

// Take in count special case, thus take the columns index

**else****if** (!has\_no\_duplicates(transpose[i], j, transpose[i][j]))

{

duplicates.insert(i);

}

}

// increase count of index in out array

**if** (max > L || all\_less)

{

out[index] += **1**;

}

**int** dup\_ind = find\_duplicate(transpose[i]);

// special case when our duplicated value, decides if the

// settement is the hottest

**if** (duplicates.count(i) && transpose[i][dup\_ind] == max)

{

out[take\_dup\_index(transpose[i], transpose[i][dup\_ind])]++;

}

}

}

**else**

{

out[**0**]++;

}

// take out the index with the biggest value

**int** biggest = give\_biggest(out);

// transer in human readable format

cout << "The warmest settlement is:**\n**";

cout << biggest + **1** << endl;

**return****0**;}

**Tes****ting**

**V****alid test cases**

1. ***test case: test1.txt***

**Input –** *minimal number of days and settlements(1)*

N = 1

M = 1

L = 20

weather11 = 1

**Output**

1

1. ***test case: test11.txt***

**Input –** *simple case 10\*10*

N = 10

M = 10

L = 30

weather11 = 1

.

.

.

weather10,10 = 10

**Output**

3

(we don’t count warmest because they are out of bound )

1. ***test case: test6.txt***

**Input –***warmest settlement has smallest maximum temperatures*

N = 3

M = 4

L = 20

weather11 = 21

weather12 = 23

weather13= 23

weather14= 21

weather21= 22

weather22= 22

weather23= 22

weather14= 22

weather31= 21

weather32= 22

weather33= 24

weather34 = 21

**Output**

2

1. ***test case: test2.txt***

**Input –***warmest settlement has smallest maximum temperatures, but all are smaller than L*

N = 3

M = 4

L = 20

weather11 = 1

weather12 = 3

weather13= 3

weather14= 1

weather21= 2

weather22= 2

weather23= 2

weather14= 2

weather31= 1

weather32= 2

weather33= 4

weather34 =1

**Output**

2

1. ***test case: test4.txt***

**Input –** *has two the warmest settlements. We take the one with smaller index*

N = 5

M = 6

L = 30

weather11 =20

.

.

.

.

weather56 =30

**Output**

2

1. ***test case: test3.txt***

**Input –** *Almost indentical to the previos one, expect here 2nd and 3rd settlements are not identical*

N = 5

M = 6

L = 30

weather11 =20

.

.

.

.

weather56 =30

**Output**

3

1. ***test case: test5.txt***

**Input –** *Big data amount. !00\*100 matrix*

N = 100

M = 100

L = 47

weather11 =46

.

.

.

.

weather100100 =43

**Output**

2

**Inva****lid test cases**

***8. test case***

**Input –** *wrong settlement type*

N = eleven11

**Output**

Error. Enter integer value

Try again:

N =

***9. test case***

**Input –** *settlement value out of bound*

N=10001

**Output**

Error. Your integer value is out of bound

Try again

Settlement

N=

***10. test case***

**Input –** *day value out of bound*

M=1001

**Output**

Error. Your integer value is out of bound

Try again

Day

M=

***11. test case***

**Input –** *wrong settlement type*

M=wrong

**Output**

Error. Enter integer value

Try again:

N =

(Same applies for L (with bounds 20 <= L <= 50))

***12. test case***

**Input –** *wrong settlement’s temperature type*

line= 20 22 22 24 asd 21

**Output**

Error. Input should be integer

Error found in line.

All temperatures in settlement should be in range 0<=Temperature<=50

Try again

line=

***13. test case***

**Input –** *wrong settlement’s temperature value*

line= 20 22 22 24 56 -23

**Output**

Error found in line.

All temperatures in settlement should be in range 0<=Temperature<=50

Try again

line=

**Fu****rther development options**

1. Data to be read from file
2. Capability to run multiple times after each other

3. Visual representation of input data, and emphasizing the result with different colors