

„Programming” C3

Made by: Cristea Andrei
Neptun code: W61RAB
E-mail: cristea.andrei997@gmail.com

Course code: IP-18fPROGEG
Teacher's name: Pluhár Zsuzsa

2020. June 05

Content

User documentation	3
Task	3
Runtime environment	3
Usage	3
Starting the program	3
Program input	3
Program output	3
Sample input and output	4
Possible errors	4
Developer documentation	6
Task	6
Specification	6
Developer environment	7
Source code	7
Solution	7
Program parameters	7
The structure of the program	8
Structure of functions	9
The algorithm of the program	10
The code	13
Testing	22
Valid test cases	22
Invalid test cases	24
Further development options	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.

Runtime 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.

Usage

Starting 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.

Program input

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

#	Data	Explanation
1.	N	Number of days weather was forecasted ($1 \leq N \leq 1000$).
2.	M	Count of settlements ($0 \leq M \leq 1000$).
3.	L	A temperature limit ($20 \leq L \leq 50$).
...	...	
$(N+1) * (M+1)$	$temperature_{N,M}$	The N^{th} settlements the M^{th} days temperature ($0 \leq temperature \leq 50$).

Program 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

Sample input and output

```
====Continuosly warmest settlement====
Please write down settlements, days and temperature limit
Separate each input by new line
Settlement
5
Day
6
Temperature Limit
30
Next, write down in each column settlement's temperature:
20 22 22 24 26 21
31 31 31 29 31 20
20 30 35 30 30 21
32 29 34 32 29 30
33 20 20 33 33 30
The warmest settlement is:
5
```

Possible 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.

Sample 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

```
====Continuously warmest settlement====
Please write down settlements, days and temperature limit
Separate each input by new line
Settlement
als
Error. Enter integer value
Try again
2300
Error. Your integer value is out of bound
Try again
Settlement
3.42
Error. Enter integer value
Try again
2Lol
Error. Enter integer value
Try again
2 3
Error. Enter integer value
Try again
5
Day
Wednesday
Error. Enter integer value
Try again
3000
Error. Your integer value is out of bound
Try again
Day
30 30
Error. Enter integer value
Try again
6
Temperature Limit
Limit
Error. Enter integer value
Try again
1001
Error. Your integer value is out of bound
Try again
Temperature Limit
51
Error. Your integer value is out of bound
Try again
Temperature Limit
30
Next, write down in each column settlement's temperature:
```

Sample 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 weather_{1..N,1..M}:

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

```
====Continuously warmest settlement====
Please write down settlements, days and temperature limit
Separate each input by new line
Settlement
5
Day
df
Error. Enter integer value
Try again
6
Temperature Limit
30
Next, write down in each column settlement's temperature:
20 22 22 24 26 wrong+here
Error. Input should be integer
Error found in line.
All temperatures in settlement should be in range 0<=Temperature<=50
Try again
55 20 23 23 1
Error found in line.
All temperatures in settlement should be in range 0<=Temperature<=50
Try again
20 22 22 24 26 21 12
31 31 31 29 31 20
20 30 35 30 30 21
32 29 34 32 29 30
33 20 20 33 33 30
The warmest settlement is:
5
```

Developer documentation

Task

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

Specification

Input: $N, M, L \in \mathbb{N}$, $\text{weather}_{1..N,1..M} \in \mathbb{N}^{N \times M}$

Output: $\text{biggest} \in \mathbb{N}$

Precondition: $N = [1..1000] \wedge M = [1..1000] \wedge L = [20..50]$

Postcondition: $\forall j(1 \leq j \leq M)$ and $\forall i(1 \leq i \leq N)$: $\text{transpose}_{j,i} = \text{weather}_{i,j} \vee$

$1 \leq \text{MaxInd} \leq N$ and $\forall j(1 \leq i \leq M)$ and $\forall i(1 \leq j \leq N)$: $\text{transpose}_{i, \text{MaxInd}} > \text{transpose}_{i,j}$
and $\text{max} = \text{transpose}_{i, \text{MaxInd}}$ and $\text{index} = j$ and $\text{has_no_duplicates}(\text{transpose}_i, j, \text{transpose}_{i,j})$
and $\text{max} > L$

$\forall j(1 \leq i \leq M)$ and $\forall i(1 \leq j \leq N)$: $\text{not has_no_duplicates}(\text{transpose}_i, j, \text{transpose}_{i,j})$:
 $\text{duplicates.insert}()$

$\forall \text{ind_dup} := \text{ind_duplicate}(\text{transpose}_i)$ and

$$\text{out}_{\text{index}} = \bigwedge_{i=1}^M 1 \text{ and } \text{out}_{\text{take_dup_index}(\text{transpose}_i, \text{transpose}_{i, \text{ind_dup}})} = \bigwedge_{i=1}^M 1$$

$\text{max} > L \text{ or all_less_than_L} \qquad \text{duplicates.count}(i) \text{ and } \text{transpose}_{i, \text{ind_dup}}$

$\forall \text{biggest} := \text{take_biggest}(\text{out})$

Definitions:

has_no_duplicates : $\text{Set}(\text{transpose}_i) = \text{transpose}_i$

give_biggest: $\forall i(1 \leq i \leq N)$: $1 \leq \text{biggest} \leq N$ and $\forall j(1 \leq i \leq N)$: $\text{out}_{i, \text{biggest}} > \text{cnt}_j$
and $\text{cnt} = \text{out}_{\text{MaxInd}}$

take_dup_index: $\exists i(1 \leq i \leq N)$: $v[i] = d$

find_duplicate: $\exists j(1 \leq j \leq N)$ and $\exists i(1 \leq i \leq N)$: $v[i] = v[j]$

all_less_than_L: $\forall i(1 \leq i \leq M)$ and $\forall j(1 \leq j \leq N)$: $\forall v[i][j] \leq L$

Developer 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.

Source code

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

File	Explanation
<i>warmest\program\warmest.exe</i>	Executable code for Windows
<i>warmest\program\warmest.out</i>	Executable code for Linux & Mac Os
<i>warmest\object\main.o</i>	Semi-compiled code
<i>warmest\main.cpp</i>	C++ source code
<i>warmest\tests\test1.txt</i>	input test file ₁
<i>warmest\tests\test11.txt</i>	input test file ₁₁
<i>warmest\tests\test2.txt</i>	input test file ₂
<i>warmest\tests\test3.txt</i>	input test file ₃
<i>warmest\tests\test4.txt</i>	input test file ₄
<i>warmest\tests\test5.txt</i>	input test file ₅
<i>warmest\tests\test6.txt</i>	input test file ₆
<i>warmest\docs\warmest_settlement.docx</i>	documentation (this file)
<i>warmest\docs\warmest_settlement.pdf</i>	documentation (this file) (pdf format)

Solution

Program parameters

Contants

`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 structure 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

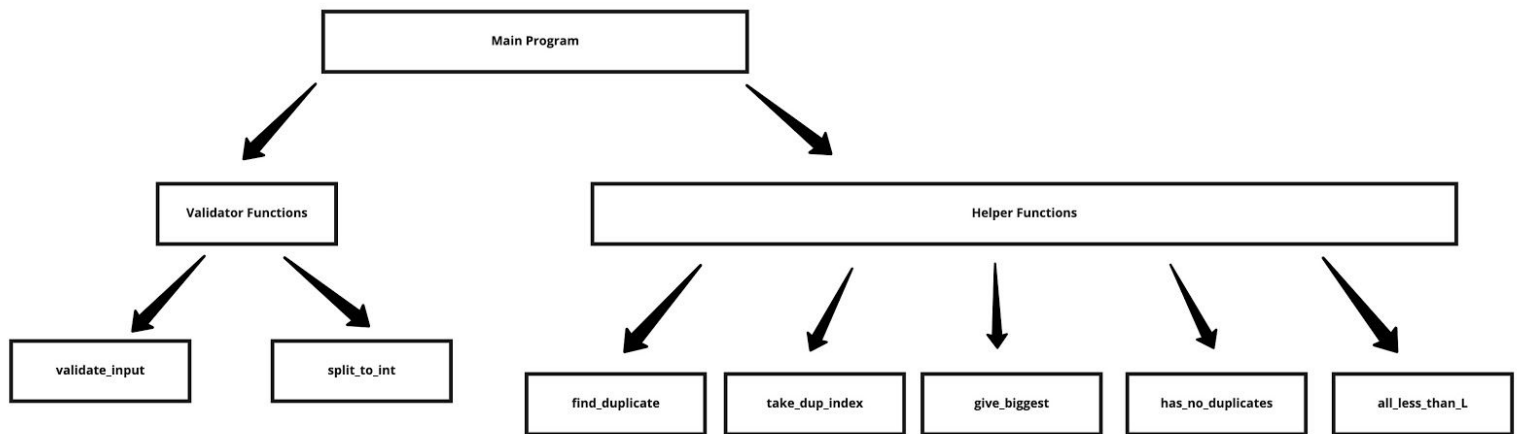
set – special type of container in C++ which has same concepts of set as in maths, part of the C++ system

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.

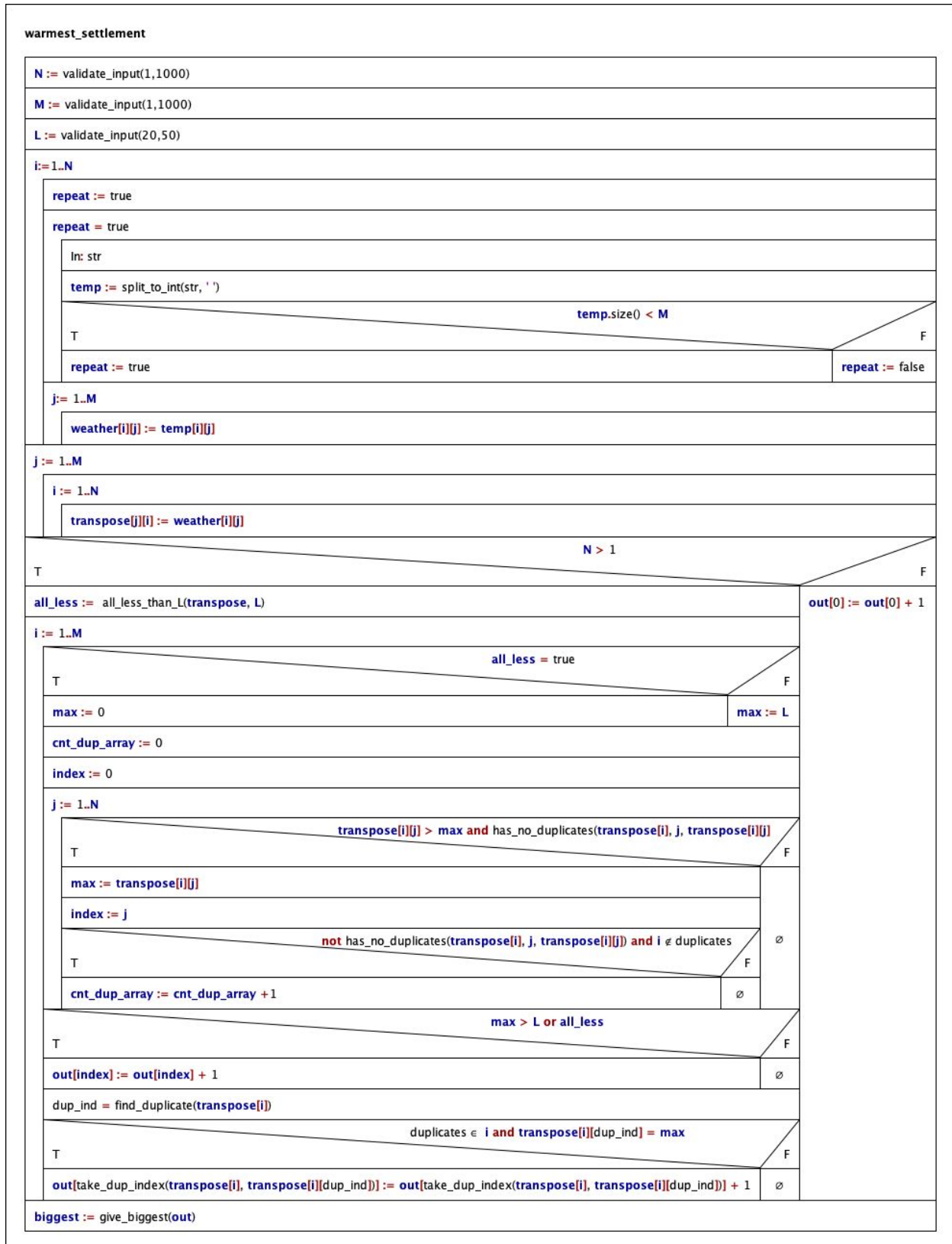
Structure of function



miro

The algorithm of the program

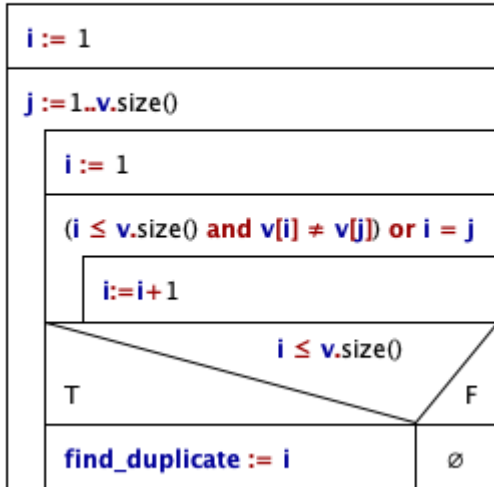
Main program:



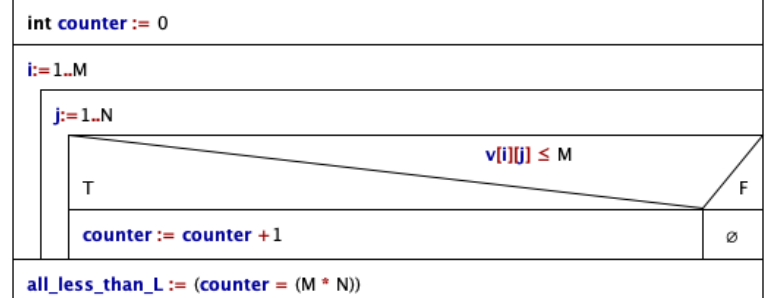
Subprograms:

Helper Functions:

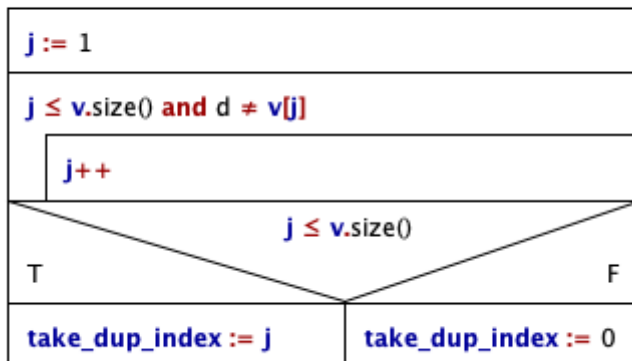
find_duplicate(v: sequence of integers)



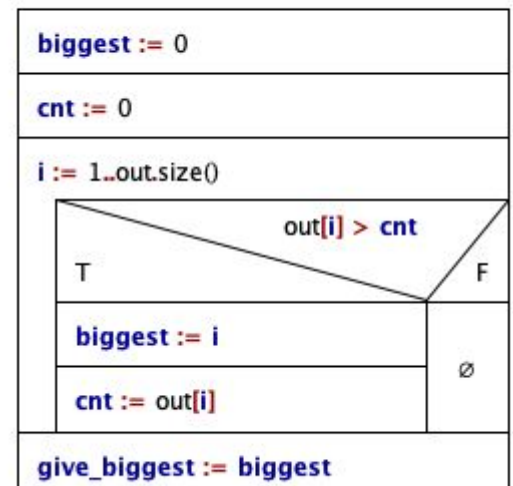
all_less_than_L(v: sequence of integer sequences, L: integer, M: integer, N: integer)



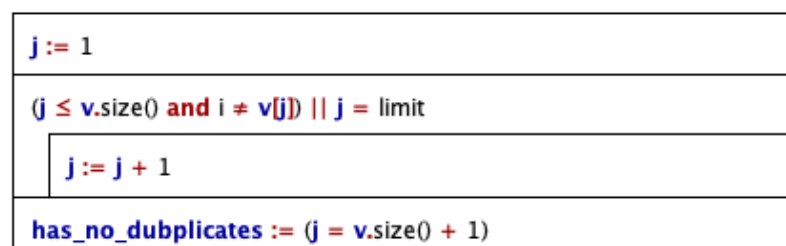
take_dup_index(v: sequence of integers, d: integer)



give_biggest(out: sequence of integers)



has_no_duplicates(v: sequence of integer, limit: integer, i: integer)



Input Validation Functions:

validate_input(low: constant template, high: constant template)

input := low - 1

input < low or input > high

In: input [low ≤ input ≤ high]

validate_input := input

split_to_int(s: constant string, delimiter: char)

cnt := 0

In: getline(tokenStream, token, delimiter)

token ≥ 0 and token ≤ 50	
T	F
cnt := cnt + 1	∅
tokens[cnt] := token	

split_to_int := tokens

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 delimiter)
{
    vector<int> tokens;

    string token;

    istringstream tokenStream(s);

    while (getline(tokenStream, token, delimiter))
    {
        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 << "====Continuously 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\nTry again\n";
```

```
const string limit_error = "Error. Your integer value is out of bound\nTry 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.\nAll temperatures in settlement should be in
range 0<=Temperature<=50\nTry 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
// settlement 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;}

```

Testing

Valid 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

2. test case: test11.txt

Input – *simple case 10*10*

N = 10
M = 10
L = 30
weather₁₁ = 1
.
.
.
weather_{10,10} = 10

Output

3
(we don't count warmest because they are out of bound)

3. 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

4. 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

5. 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

6. test case: test3.txt

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

$N = 5$

$M = 6$

$L = 30$

weather₁₁ = 20

.

.

.

.

weather₅₆ = 30

Output

3

7. test case: test5.txt

Input – Big data amount. 100*100 matrix

N = 100

M = 100

L = 47

weather₁₁ = 46

.
. .
. .
. .

weather₁₀₀₁₀₀ = 43

Output

2

Invalid 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 $20 \leq L \leq 50$))

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 $0 \leq \text{Temperature} \leq 50$

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 $0 \leq \text{Temperature} \leq 50$

Try again

line=

Further 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