**Міністерство освіти і науки України**

**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

**Факультет інформатики та обчислювальної техніки**

**Кафедра обчислювальної техніки**

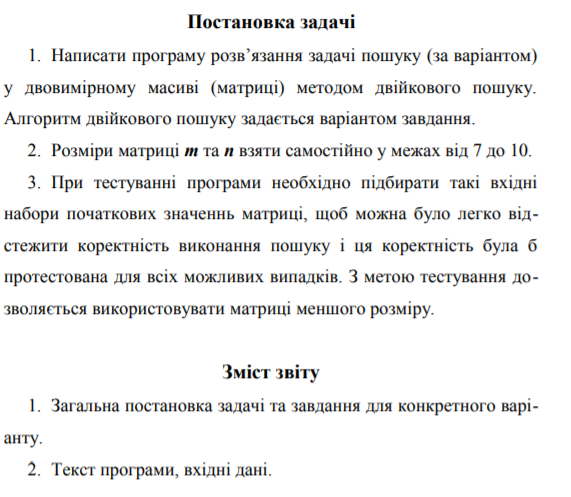
**Лабораторна робота №2.1**

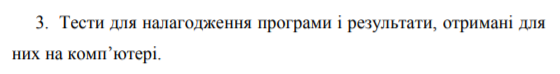
з дисципліни  
«Алгоритми двійкового пошуку»

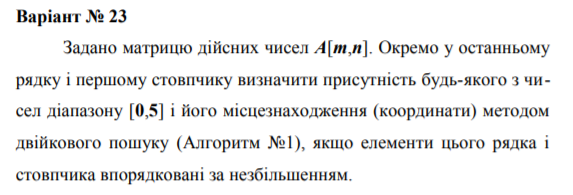
Виконав:  
студент групи ІО-04  
Федорко Андрій Петрович  
номер у списку групи: 23

Перевірила:  
Сергієнко А.А.

**Завдання:**







Текст програми  
<https://github.com/Ap3lsin4k/find-thief-in-the-city>

//main.c

#include <stdio.h>  
#include "implementation.h"  
  
int main() {  
 double const\_array[7][9] = {  
 {90, 9, 3.2, 9.2, 32.2, -6, 20, 99, 35},  
 {75.7, 7.7, 2.5, 7.0, 10, -6.2, 19.8, 48, 29},  
 {31, 7, 2, 5.2, 5, -6.3, 17, 43.3, 21.9},  
 {30, 7, 2, 5.1, 5, -7, 16.3, 41.3, 11},  
 {24, 4, 2, 2.3, 2.3, -8, 10, 12, 1},  
 {22.3, 2.3, 1.25, 2, -1, -11, 10, -12, 1},  
 {10, 1, 0.5, 0, -8.08, -11.99, -15, -22, -23}  
 };  
  
 FIND\_THIEVES\_IN\_CITY(const\_array, 7, 9);  
 if (result.found)  
 printf("A thief %lf has been found at %dx%d (indexing from 1 not 0)", const\_array[result.row][result.column], result.row+1, result.column+1);  
 else  
 printf("All inhabitant are innocent");  
  
 return 0;  
}

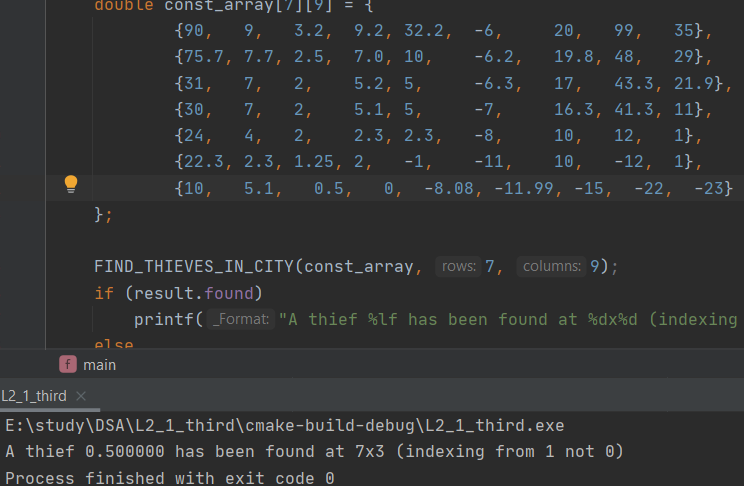
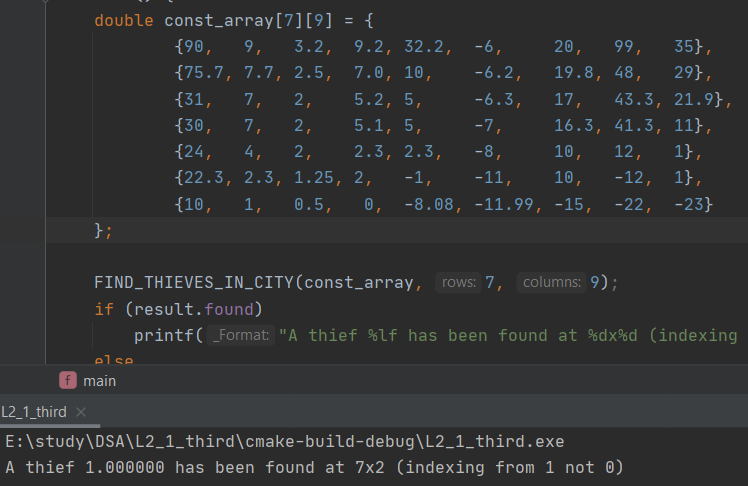
//implementation.c – first programme

#include <stdbool.h>  
#include "implementation.h"  
  
//  
// Created by Andy on 2020-11-29.  
//  
  
bool is\_wanted(double thief)  
{  
 return 0 <= thief && thief <= 5;  
}  
  
bool less\_than\_wanted(double thief)  
{  
 return thief < 0;  
}  
  
bool greater\_than\_wanted(double thief)  
{  
 return thief > 5;  
}  
  
void imprison(struct position\* pos, unsigned short row, unsigned short col)  
{  
 pos->found = true;  
 pos->row = row;  
 pos->column = col;  
}  
  
void binarySearchInLastRow(double \*const \*city, unsigned short columns, struct position \*pos, unsigned short last\_row\_index);  
void binarySearchInFirstColumn(double \*const \*city, struct position \*pos, unsigned short last\_row);  
  
struct position find\_thieves(double \*\*city, unsigned short rows, unsigned short columns)  
{  
 struct position pos;  
 pos.found = false;  
 unsigned short last\_row = rows - 1;  
  
 binarySearchInLastRow(city, columns, &pos, last\_row);  
  
 if (!pos.found)  
 binarySearchInFirstColumn(city, &pos, last\_row);  
  
 return pos;  
}  
  
void binarySearchInFirstColumn(double \*const \*city, struct position \*pos, unsigned short last\_row) {  
 unsigned short topPivot = 0;  
 unsigned short bottomPivot = last\_row;  
 unsigned short middle;  
  
 while (topPivot <= bottomPivot)  
 {  
 middle = (topPivot + bottomPivot) / 2;  
 if (is\_wanted(city[middle][0]))  
 {  
 imprison(pos, middle, 0);  
 return;  
 }  
 else if (greater\_than\_wanted(city[middle][0]))  
 {  
 topPivot = middle + 1;  
 }  
 else if (less\_than\_wanted(city[middle][0]))  
 {  
 if (middle > 0) bottomPivot = middle - 1;  
 else { return; }  
 }  
 }  
}  
  
void binarySearchInLastRow(double \*const \*city, unsigned short columns, struct position \*pos, unsigned short last\_row) {  
 unsigned short leftPivot = 0;  
 unsigned short rightPivot = columns - 1;  
 unsigned short middle;  
  
 while (leftPivot <= rightPivot)  
 {  
 middle = (leftPivot + rightPivot) / 2;  
 if (is\_wanted(city[last\_row][middle]))  
 {  
 imprison(pos, last\_row, middle);  
 return;  
 }  
 else if (greater\_than\_wanted(city[last\_row][middle]))  
 {  
 leftPivot = middle + 1;  
 }  
 else if (less\_than\_wanted(city[last\_row][middle]))  
 {  
 if (middle > 0) rightPivot = middle - 1;  
 else return;  
 }  
 }  
}

}

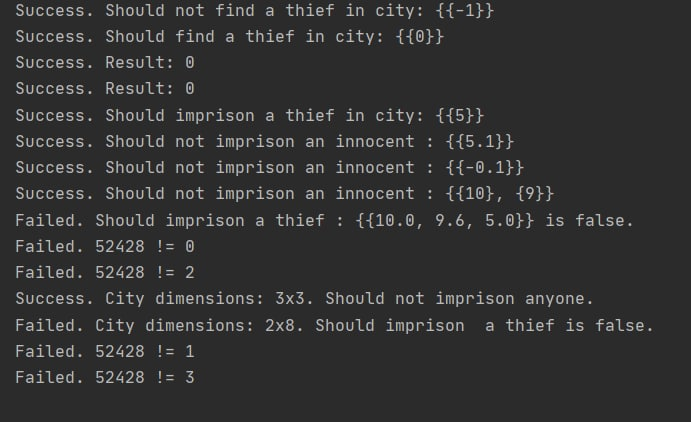
//implementation.h

//  
// Created by Andy on 2020-11-29.  
//  
#include <stdbool.h>  
#include <stdlib.h>  
  
#ifndef L2\_1\_THIRD\_IMPLEMENTATION\_H  
#define L2\_1\_THIRD\_IMPLEMENTATION\_H  
  
#define CREATE\_ARRAY(name, rows, columns) \  
 double \*\*name; \  
 name = malloc(rows \* sizeof \*name); \  
 for (int i=0; i<rows; i++) \  
 name[i] = malloc(columns \* sizeof \*name[i]);  
  
#define FILL\_ARRAY(out\_array, const\_expression, rows, columns) \  
 double const\_array[rows][columns] = const\_expression; \  
 for (unsigned short i=0; i < rows; ++i) \  
 for (unsigned short j=0; j<columns; ++j) \  
 out\_array[i][j] = const\_array[i][j];  
  
#define DEALLOCATE\_ARRAY(name, rows, columns) \  
 for (unsigned short i=0; i<rows; i++) \  
 free(name[i]); \  
 free(name);  
  
#define CONTROLLER\_FIND\_THIEVES(matrix\_brackets, rows, columns) \  
 CREATE\_ARRAY(city, rows, columns); \  
 FILL\_ARRAY(city, matrix\_brackets, rows, columns); \  
 \  
 struct position result = find\_thieves(city, rows, columns); \  
 DEALLOCATE\_ARRAY(city, rows, columns); \  
  
  
#define FILL\_MATRIX\_FROM\_CONSTANT\_MATRIX(out\_matrix, const\_matrix, rows, columns) \  
 for (unsigned short i=0; i < rows; ++i) \  
 for (unsigned short j=0; j<columns; ++j)\  
 out\_matrix[i][j] = const\_matrix[i][j];  
  
#define FIND\_THIEVES\_IN\_CITY(my\_city, rows, columns) \  
 CREATE\_ARRAY(dynamic\_matrix\_city, rows, columns); \  
 FILL\_MATRIX\_FROM\_CONSTANT\_MATRIX(dynamic\_matrix\_city, my\_city, rows, columns); \  
 struct position result = find\_thieves(dynamic\_matrix\_city, rows, columns); \  
 DEALLOCATE\_ARRAY(dynamic\_matrix\_city, rows, columns);  
  
struct position  
{  
 bool found;  
 unsigned short row, column;  
};  
  
struct position find\_thieves(double \*\*city, unsigned short rows, unsigned short cols);  
  
  
  
  
  
  
#endif //L2\_1\_THIRD\_IMPLEMENTATION\_H

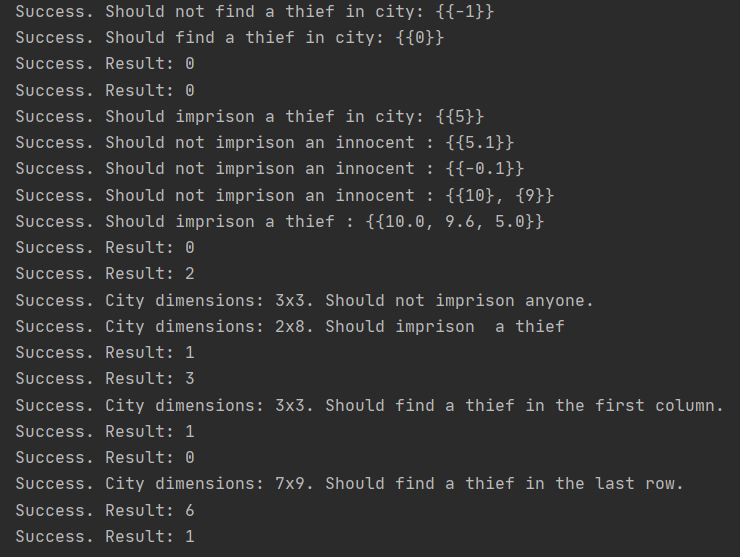
Manual testing  


UNIT testing

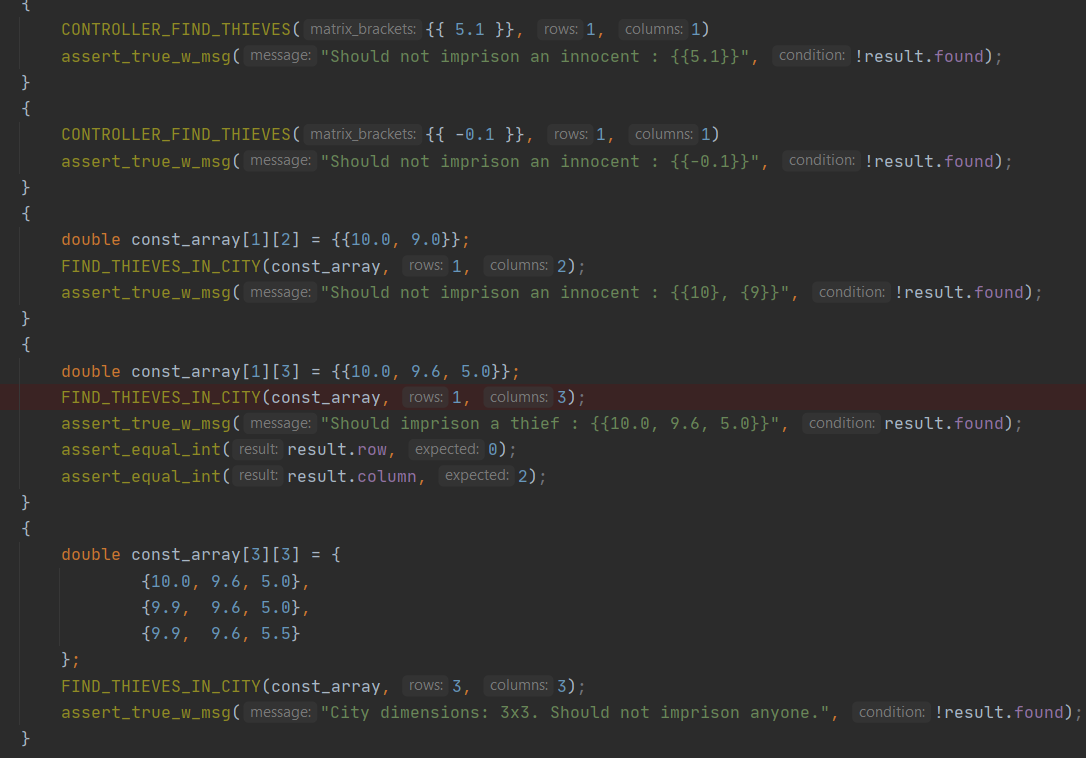
Під час переходу від лінійного пошуку до бінарного тести впали



Але помилка була в знаку > або <



Отже, автоматичні тести дозволяють швидко знайти при яких вхідних данних програма не працює, і полегшує процес debugging та refactoring.

  
Деякі з UNIT tests.