04JCJLZ - COMPUTER SCIENCES - 2015/2016

Laboratory 9

Objectives

- Realizing programs with complex data, using files

Technical content

- Using matrixes of type integer and character
- Using sets of "parallel" arrays
- Reading text files.

Preferably to be solved in the laboratory

Exercise 1. A matrix of characters schematically represents a swamp. The swamp is composed of muddy areas, represented with the '.' character, and rocky areas, represented with the '*' character. The matrix size can be defined using **#define**, however it should be no greater than 25 rows and 80 columns.

Example of swamp:

```
**.*.*....*
..*.*...**.
*...*.*....
.*.*.*.*.*.
..*.*...*.*
```

Realize a program that searches a path in the swamp, from left to right, without jumps, only including consecutive rocky areas. Suppose that each rocky area can have at most one other rocky area on its right (there are no branches), i.e., either on the same row, or in the previous row, or in the following one. The program shall print the row sequence of the path (the columns are implicit – there shall be a rocky area for each column), or report that no path exists.

For example:

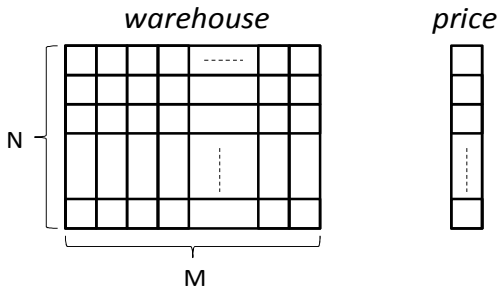
```
0   **.*.*....*
1   ..*.*...**.
2   *...*.*....
3   .*.*.*.*.*.
4   ..*.*...*.*
```

Path: 2 3 4 3 4 3 2 3 4 3 4

Hint: in a preliminary version, use a predefined matrix of strings and test the program; then modify the program and read the swamp from the keyboard input (in the future it would be possible to read it from a file).

Exercise 2. Write a C program that is able to manage a price list, i.e., a program which can manage a list of products and the corresponding prices in euros. The program shall use a N×M matrix of characters called *warehouse* in order

to store the product names (at most N products), and an array of N real numbers called *price*, where to store the prices (the value stored in the *i*-th element of *price* corresponds to the price of the *i*-th product, which name is stored in the *i*-th row of *warehouse*). The logical representation of the data is the following:



In the *price* array, the value -2 indicates that the line is empty (i.e., no product exists in the corresponding row of *warehouse*), while a positive value indicates the price of a valid product, which name is stored in the corresponding row of *warehouse*. When the program starts, all the elements of *price* are initialized to -2 (thus, the list of products is empty).

The program proposes to the user a menu list of possible operations. The user decides which operation to be executed by selecting the index number associated with the interested operation. The available operations are:

1. Insertion of a new product and the associated price
2. Print the current list (a table with product names and prices)
3. Exit the program.

The first two operations must be realized using the following functions, which prototypes will be given:

- *insert_product*: it is a function that is able to insert a new product and its price (the name of the product is at most M-1 characters long). The function returns 1 if the new product is not already in the product list and is inserted successfully; it returns 0 if the new product is already in the list (i.e., the product has been inserted previously); it returns 2 if the list is full (consequently, it is not possible to insert the new product). If the product is not already in the list, the function must insert its name into the first empty element of *warehouse* and the associated price into the corresponding element of *price*.

```
int insert_product(char warehouse[][M], float price[], int n,
                   char new_product[], float price_new_product);
```

The *n* parameter represents the maximum number of rows in the *warehouse*, which also corresponds to the number of elements in *price* (in our case, the function shall be invoked passing the value N).

- *print_all*: it is a function that prints to the screen the content of the list (the table of the product names and prices). In addition, the function returns two values (with two parameters that pass the address of the matrix and the array): the average and the maximum price in the product list. The two returned values should be displayed on the screen.

```
void print_all(char warehouse[][M], float price[], int n,
                                    float *avg, float *max);
```

Note: the exercise suggests to use *price* in a dual fashion: to store the price of the products, and as *flag* of empty positions. Take this into account when to search for the name of a product, when to print, etc.

To be solved at home

Exercise 3. Write a program that reads from a file, whose name is to be entered from the keyboard, some train information. In each line, the file contains the following information (each field is no longer than 20 characters and has no spaces):
<*departure_station*> <*departure_time*> <*arrival_station*> <*arrival_time*>

After that, the program receives from keyboard the name of a station: it shall calculate and print the number of trains departing from and arriving to that station (if included in the list)

Exercise 4. Extend the program realized in the exercise 2 by adding two further operations:

4. Update the price of a product
5. Remove a product.

The two operations should be realized by using the following functions:

- *update_product*: it is a function that updates the price of a specific product. The function receives the name of the product to be updated and its new price. It returns 1 if the update is successful, 0 if the product is not in the list.

```
int update_product(char warehouse[][M], float price[],
                int n, char product[] , int new_price);
```

- *remove_product*: it is a function that removes a product from the list. The function returns 1 if the removal is successful, 0 if the product does not exist. Furthermore, the function must reset the price of the eliminated product to -2 in the price list.

```
int remove_product(char warehouse[][M], float price[],
                            int n, char old_product[]);
```

Exercise 5. Implement a C program and simulates the movement of a mouse.
There is a matrix Q of dimension N*N, which describes the characteristics of a certain area: in each location (x,y) of the matrix Q stores the proportion of square surface in the coordinate (x,y) of the area. The mouse starts from an initial position (x0,y0), and the program should print out the coordinates of all the squares the mouse touched when it moves following the rules as follows:

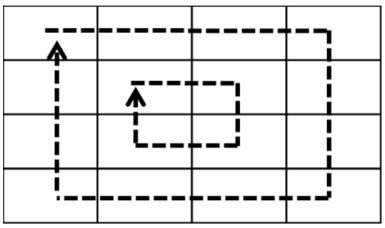
- The mouse moves to one of the 8 surrounding squares at each step
- The mouse chooses the square with the largest proportion of square surface among the 8 ones to move to
- The mouse stops if all the 8 surrounding squares have lower or equal proportion of square surface with respect to the square it is currently on

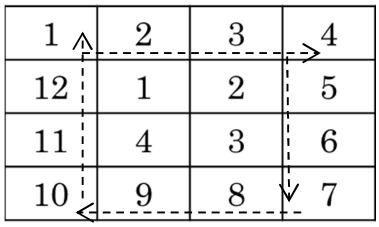
Example: Assuming we have the following Q matrix of size 10*10, and the mouse starts at the location (3,7). Then the mouse will move following: (3,7) (4,8) (5,8) (6,8) (7,9). And it will stop at location (7,9).

	1	2	3	4	5	6	7	8	9	10
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3	0.1
3	0.1	0.2	0.3	0.3	0.4	0.5	***0.5***	0.5	0.5	0.1
4	0.1	0.2	0.3	0.3	0.5	0.5	0.5	0.7	0.7	0.1
5	0.1	0.2	0.4	0.4	0.5	0.7	0.7	0.8	0.7	0.1
6	0.1	0.2	0.4	0.4	0.5	0.7	0.8	0.9	0.8	0.1
7	0.1	0.2	0.4	0.4	0.5	0.7	0.8	0.9	**1.4**	0.1
8	0.1	0.2	0.4	0.4	0.5	0.7	0.8	0.9	1.2	0.1
9	0.1	0.2	0.4	0.4	0.5	0.7	0.8	0.9	1.1	0.1
10	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

When testing the program at first, you can initialize the matrix and start point statically in your code. And afterwards, change the program to load the matrix from file and read the start point location from keyboard.

Exercise 6. Write a program that is able to fill a N×N square matrix of integers (with N greater or equal to 4 as a constant defined by the #define statement) according to the pattern of concentric circles. For each circle, we start from the top left cell and fill the circle with progressively increasing numbers starting from 1, as shown in the following figures:





Continue this process until all the columns and rows are filled properly.