Practice 6: Arrays (1 session)

Week 18tn november 2024

Exercise 1

Write a program in C that asks the user for a text of maximum 200 characters and prints out on the screen the number of words, the number of vowels and the number of consonants in it. At least we must implement one module responsible for asking for the text and another module receiving the text and returning the number of words, the number of vowels and the number of consonants.

Example of operation

```
$ ./exercise1

Enter a text of maximum 200 characters: hello cruel world
Text: hello cruel world
* Words: 3
* Vowels: 5
* Consonants: 10
```

Exercise 2

Write a program in C that asks the user for a word (maximum 20 letters) and indicates if it is a palindrome or not. You need to implement a module to check if the word is palindrome that will receive the word and return `true` or `false`.

Example of operation

```
$ ./exercise2

Write a word: recognize
The word recognize is NOT palindrome

$ ./exercise2

Write a word: sees
The word week is a palindrome
```

Exercise 3

Write a C program that fills in an array with 10 random numbers between 1 and 50. The program will print out the array and the largest and smallest numbers in that array. The following modules must be implemented:

- fillArray. Module that receives the array and fills it in with random numbers between 1 and 50.
- printArray. Module that receives the array and prints it out.
- minANDmax. Module that receives the array and returns the minimum and maximum values.

Example of operation

```
$ ./exercise3

Vector: 21, 25, 48, 3, 5, 4, 17, 24, 15, 32
* Min: 3
* Max: 48
```

Exercise 4

Write a program in C that asks the user for integer numbers until the user enters 0 or the user has entered 20 numbers. The program will then indicate how many numbers have been entered and print out the entered numbers in increasing order (from the smallest to the largest). It must be implemented in a modular way: a module will ask the user for the numbers and return the array of numbers and the total amount of numbers; another module will receive the array and the amount of numbers stored in it and will sort the array; finally, there will be a module that will receive the array and the amount of numbers stored in it and will print the array on the screen.

Example of operation

```
$ ./exercise4

Enter an integer number (0 to exit): 5
Enter an integer number (0 to exit): 3
Enter an integer number (0 to exit): 28
Enter an integer number (0 to exit): 54
Enter an integer number (0 to exit): 23
Enter an integer number (0 to exit): 17
Enter an integer number (0 to exit): 8
Enter an integer number (0 to exit): 0
Ordered numbers: 3, 5, 8, 17, 23, 28, 54
```

Exercise 5

Write a C program that implements the addition of matrice of integers. The dimensions of the matrices to be added will be asked to the user when starting the program, and cannot be larger than 10x10. Then, the user will be asked for the elements of each matrix and the sum will be performed, displaying the resulting matrix. At least the following modules must be implemented:

- askDimensions. It will ask the user for the dimensions (rows and columns) of the matrices, checking that they are values less than or equal to 10, and will return them.
- askMatrixElements. It receives a matrix and its number of rows and columns, and fills it with the elements that the user enters.
- printMatrix. It receives a matrix and its number of rows and columns, and prints it out on screen.
- addMatrices. It receives the two matrices to be added, the resulting matrix and the number of rows and columns. The module will add the two matrices and save the result in the resulting matrix.

Example of operation

```
$ ./exercise5

Enter the number of rows (1-10): 12
Enter the number of rows (1-10): 2
Enter the number of cols (1-10): 3
Operation: A + B
Enter the elements of the matrix A:
Enter the 2 elements of row 0 separated by spaces: 5 -2 0
Enter the 2 elements of row 1 separated by spaces: 1 -1 3
Enter the elements of the matrix B:
Enter the 2 elements of row 0 separated by spaces: 0 -3 5
Enter the 2 elements of row 1 separated by spaces: 3 7 -4
Matrix A:
   5  -2   0
   1  -1   3
Matrix B:
   0  -3   5
   3   7  -4
The result of A + B is:
   5  -5   5
   4   6  -1
```

Exercise 6

Write a C program that fills a 5x10 matrix with random integers between 1 and 100. The program will then find the maximum and minimum of each row and each column and save them in two matrices: a 5x2 matrix for the rows and another 10x2 matrix for the columns (Use: for these two matrices, the first column will store the minimum value and the second the maximum value of the corresponding row/column). Finally, the program will print out the matrix and the maximums and minimums of each row and each column. At least the following modules must be implemented:

- fillMatrix. It receives the 5x10 matrix and fills it with random integers between 1 and 100, inclusive.
- searchMinMaxRows. It receives the 5x10 matrix of numbers and prints out the minimum and maximum of each row.
- searchMinMaxColumns. It receives the 5x10 matrix of numbers and prints the minimum and maximum of each column.
- printMatrix. It receives the array of numbers and prints it on screen.

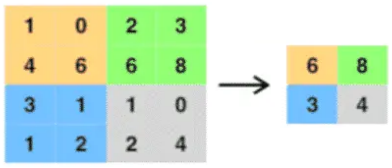
Example of operation

```
$ ./exercise6

Matrix 5x10:
   3  48  38  90  15  44 100  51  21  50
  20  34  97  25  43  66   5  87  14  80
  29  11  63  37  24  96  85  58  60  50
   8  14  49  97   3  64  93  54  66  13
  55  85  46  51  62  40  17  18  27  82
```

```
Row mins and maxs:
* ROW 1 => Min:   3 - Max: 100
* ROW 2 => Min:   5 - Max:  97
* ROW 3 => Min:  11 - Max:  96
* ROW 4 => Min:   3 - Max:  97
* ROW 5 => Min:  17 - Max:  85

Mins and maximums of the columns:
* COL 1 => Min: 3 - Max: 55
* COL 2 => Min: 11 - Max: 85
* COL 3 => Min: 38 - Max: 97
* COL 4 => Min: 25 - Max: 97
* COL 5 => Min: 3 - Max: 62
* COL 6 => Min: 40 - Max: 96
* COL 7 => Min: 5 - Max: 100
* COL 8 => Min: 18 - Max: 87
* COL 9 => Min: 14 - Max: 66
* COL 10 => Min: 13 - Max: 82
```

Exercise 7

In a convolutional neural network (CNN), reduction or *pooling* layers are applied on a data matrix. One way to apply this reduction is to calculate the maximum of a fixed region of the matrix as a representative of the region (*MaxPooling*), which reduces the dimensions of the data matrix. The size of the region is determined by the *stride*. See below an example of the process for a 4x4 matrix with *stride*=2:



Write a program in C that asks the user for the dimension `d` (between 4 and 32) of a `dxd` integer matrix on which a reduction process (with *stride*=2) will be applied. The program will print out the generated integer matrix and the matrix resulting from the reduction. At least the following modules must be implemented:

- orderDimension. It will ask the user for the dimension of the integer matrix and return it. This module must check that the dimension is a power of 2. As long as the dimension is not correct, it will continue to request it.
- fillMatrix. It receives the matrix of integers to be filled and its dimension, and fills it with random numbers between 1 and 20.
- applyReduction. It will receive the integer array, its dimension and return the reduced array. The matrix in which the result of the reduction will be saved can be received as a second argument or, if you want to increase the difficulty of the exercise, you can use the same array of integers to store the reduced matrix in it. Remember that the dimension of the resulting matrix will be half of the original.
- printMatrix. It receives the array of integers to be filled and its dimension, and prints it on the screen.

Example of operation

```
$ ./exercise7

Matrix dimension (power of 2 between 4 and 32): 4
MATRIX:
  4  4 20  3
 15 17  8  7
 18 16  4  8
 16  3 14 17

Reduction:
 17 20
 18 17

$ ./exercise7

Matrix dimension (power of 2 between 4 and 32): 8
MATRIX:
  2 20 18 12 18 11 16  8
 14  1 15 14 12  1 12  1
 20  9  8 19  7  9 11 15
  7 13  2 17  1 17 12  2
 16  9 13  5 19 20  4 13
 20 19 18  3 19  9  3 10
 17  3  1 15 11 11  9 17
 16  2  6 16 10 17 17  5

Reduction:
 20 18 18 16
 20 19 17 15
 20 18 20 13
 17 16 17 17
```