

# Practice Questions for LAB 2 (Oct. 1-19)

## Functions and Arrays

### General Requirements on Your Programs

- A. Your programs should be written in a good programming style, including instructive comments and well-formatted, well-indented code. Use self-explanatory names for variables as much as possible. (~5% of the mark)
- B. When outputting the results, include in the output an explanatory message. When inputting values display a message prompting the user to input the appropriate values. (~10% of the mark)

### Lab Questions

Assume the character code used is ASCII.

**You are allowed to use from the C standard library only functions for input and output (e.g. *printf*, *scanf*, *getchar*) and math library functions. The use of global variables is not permitted.**

1. Consider representing vectors with  $n$  floating point components using arrays. Develop a library for vector operations, that includes the following functions.
  - A function `print_vector()` to print the vector components, on a single line, separated by commas. The function prototype has to be:
    - `void print_vector(double array[], int size);`Parameter size represents the size of parameter array and the vector dimension.
  - A function `add_vectors()` to add two vectors of the same size, with prototype:
    - `void add_vectors( double vector1[], double vector2[], double vector3[], int n);``vector3` should store the sum of `vector1` and `vector2`. You may assume that all three arrays have the size equal to  $n$ , which equals the vector dimension. (In other words, assume that the calling function ensures that the arrays passed in satisfy this condition.)
  - A function `scalar_prod()` that returns the scalar product of two vectors of the same dimension. You may assume that the passed in arrays have the same size.
  - A function `norm2()`, which returns the L2 norm of a vector. The L2 norm is defined as the square root of the scalar product of the vector with itself. Function `norm2()` should call function `scalar_prod()`.

Write a program to test this library. You **are allowed** to use math library functions.

**Attention:** When you pass an array (which is not a string) to a function, you also need to pass to the function the size of the array.

**Note:** Consider vectors  $x=(x(0), x(1), \dots, x(n-1))$  and  $y=(y(0), y(1), \dots, y(n-1))$ .

The **sum** of  $x$  and  $y$  is the vector  $z=(z(0), z(1), \dots, z(n-1))$ , where  $z(i)=x(i)+y(i)$  for every  $0 \leq i < n$ .

The **scalar product** of  $x$  and  $y$  is the value  $x(0)y(0)+x(1)y(1)+\dots+x(n-1)y(n-1)$ .

**Example:** Assume  $n=3$  and vectors  $x=(2,4,6)$  and  $y=(0,1,2)$ . Then the sum of vectors  $x$  and  $y$  is the vector  $sum=(2,5,8)$ . The scalar product of vectors  $x$  and  $y$  is the number  $0+4+12=16$ .

2. Write a function to determine if an array of integers is sorted.
3. Write a function which determines how many times a particular integer appears in an array of integers.

4. Write a function with prototype

- `void change(int x[], int n)`

that is passed an array `x` of integers and its size `n`. The function changes the order of the array elements such that all the zeros will be placed at the end of the array, while the non-zeros will be placed at the beginning of the array in the same relative order. For example, if the array initially contains

0, -1, 2, 0, 7, 6, -8, 0,

then after the function executes the array contains

-1, 2, 7, 6, -8, 0, 0, 0.

**Note:** If necessary you may allocate another array for temporary storage.

5. A diagonally dominant matrix is a matrix  $A$  such that for each row, the absolute value of the diagonal element on that row is strictly larger than the sum of the absolute values of all other elements in the row. That is, for each row  $i=0,1,\dots, n-1$ , the following holds:

$$|a_{ii}| > \sum_{j=0, j \neq i}^{n-1} |a_{ij}|$$

Write a function `is_diag_dom()` that determines if an  $N$ -by- $N$  matrix `mat` is diagonally dominant (it returns 1 if the matrix is diagonally dominant and 0 otherwise). The function prototype has to be

- `int is_diag_dom( int mat[ ][N])`.

You may use the function `fabs()` with prototype

- `double fabs(double x)`,

from the C standard math library, which returns the absolute value of `x`. Write a program to test this function. Note that  $N$  represents a constant. To set a value to  $N$  use the define directive at the beginning of the file (E. g.: `#define N 20` replaces  $N$  by 20 all over the file, except for occurrences of  $N$  inside a string or a variable name). Write a program to test the function.

6. Write a function which prints all elements of a square matrix in a diagonal scan order, starting at the top left corner. For instance, for the following matrix

```
1 12 13 49
5 16 17 81
9 10 11 20
2 45 19 14
```

the output has to be: 1 5 12 9 16 13 2 10 17 49 45 11 81 19 20 14

You have to decide the function prototype.

Write a program to test the function.

7. In linear algebra, a Toeplitz matrix or diagonal-constant matrix, is a matrix in which every descending diagonal from left to right is constant (in other words, has all its elements equal). Such matrices are encountered in applications in signal processing and communications. For instance, the following 4-by-5 matrix is a Toeplitz matrix:

```
9 2 1 0 4
7 9 2 1 0
3 7 9 2 1
5 3 7 9 2
```

Write a function with prototype\\

- `int is_toeplitz(int a[][N], int m)`,

which determines if the  $m$ -by- $N$  matrix `a` is a Toeplitz matrix or not. The function has to return 1 if the answer is positive and 0 otherwise. Note that  $N$  is a symbolic constant defined at the beginning of the C

file.

8. Write a C function with prototype

- `void letter_freq(const char word[], int freq []);`

This function computes the number of appearances of each letter in the string `word` and stores them in array `freq` of size 26. The letters are the 26 letters of the Latin alphabet whose ASCII values are in the range 97-122 for the lower case letters, and in the range 65-90 for the uppercase letters. ***You must account for uppercase and lowercase letter variants, which should be counted together.*** The counts have to be stored in array `freq` in alphabetical order of letters, which corresponds to the increasing order of their ASCII values. Specifically, `freq[0]` should store the count of 'A' and 'a', `freq[1]` should store the count of 'B' and 'b', and so on. This function has also to print the counts indicating each letter and its count on a separate line, as follows:

The count of 'A' and 'a' is ...

The count of 'B' and 'b' is ...

...

Write a program to test the function.

**Hint:** If variable `x` of type `char` represents a lower case letter, then the corresponding index in the array equals the integer value of `x-'a'`. If `x` is an upper case letter, then the index in the array equals `x-'A'`.

9. Write a function with prototype

- `void string_copy(const char source[], char destination[], int n);`

This function copies string `source` to string `destination`. Parameter `n` represents the size of array `destination`. If the latter array is not sufficiently large to hold the whole source string then only the prefix of the string which has room in the latter array should be copied. Note that after copying, the null character should also be included to mark the end of string `destination`.

Write a program to test your functions.

***You are not allowed to use any function declared in `string.h`.***

**You may write a function which returns the length of a string and use it if you need it. Recall that a string is a char array with the null character marking the end. The length of the string is the number of characters in the array appearing before the null character.**

10. Implement a string processing function with the prototype

- `void my_strcat( const char str1[], const char str2[], char str3[], int n);`

The function concatenates strings `str1` and `str2` and stores the new string in `str3` if the array `str3` is large enough. Otherwise, an error message signaling this should be printed. Note that the parameter `n` equals the size of the array `str3`.

Write a program to test the function.

11. Implement a string processing function with the prototype

- `int my_strcmp( const char str1[], const char str2[]);`

The function compares the string `str1` with the string `str2` according to the alphabetical order. The function returns 0, -1 or 1 if `str1` is equal to, less than or greater than `str2`, respectively.

Write a program to test the function.

12. Write a function which determines whether a string `s` is a proper prefix of another string `p`. `s` is a prefix of `p` if `s` appears at the beginning of `p`. `s` is a proper prefix if it is nonempty and not equal to `p`. Examples: "ba" is a proper prefix of "base", "bac" is not a prefix of "base". For simplicity, you may assume that all strings contain only lowercase letters. You have to decide the appropriate prototype for this function.

13. A sparse vector is a vector whose most components are zero. To store a sparse vector efficiently it is enough to store only its non-zero components and their index (position in the vector). The components of a vector are indexed starting from 0, like in C arrays. Precisely, to store a sparse vector with **n** components, only **k** of which are non-zero, we can use two arrays: **val** and **pos**, each of size **k**. For example, if the sparse vector **x** with 8 components is the following

0   0   23   0   -7   0   0   48

then **k=3** and

**val** contains      23       -7       48

**pos** contains      2        4        7

Notice that the elements of array **pos** are in increasing order. **We will assume that each vector contains at least one non-zero element.**

Write a function `efficient()` with prototype

- `void efficient( const int source[], int val[], int pos[], int size)`

which computes the efficient representation of vector `source`, by filling the arrays `val` and `pos`. Parameter `size` represents the number of components of vector `source` (i.e., the size of the array). Assume that the size of arrays `pos` and `val` equals the number of non-zero values of vector `source`.

Additionally, write a function `reconstruct()` with prototype

- `void reconstruct( int source[], int m, const int val[], const int pos[], int n)`

which reconstructs vector `source` from the efficient representation stored in arrays `val` and `pos`. Parameter `n` represents the size of arrays `val` and `pos`. Parameter `m` represents the size of array `source`, which equals the dimension of the vector.

Write a program to test the two functions.

14. Consider the efficient representation of sparse vectors as in question 6. Write a function with prototype

- `void addEff( int val1[], int val2[], int val3[],int pos1[], int pos2[],int pos3[], int k1, int k2)`

where `val1`, `pos1` and `val2`, `pos2` represent two sparse vectors of integers, stored efficiently. `k1` is the number of non-zero elements of vector 1 and `k2` is the number of non-zero elements of vector 2. Function `addEff()` has to add the two vectors and store the result in efficient representation as well, using `val3`, `pos3`. Assume that the size of arrays `val3` and `pos3` equals the number of non-zero elements in the sum vector, but the function does not know this number. **The function is not allowed to allocate any array**, in other words only a constant amount of variables may be allocated during the function execution. **Note: Pay attention to the case when two non-zero elements sum up to 0.** You may assume that the two vectors, as well as their sum, are not equal to 0.