COMPUTER SCIENCE

Practice 4.

Basic programming in C.

GIEyAI

Fall semester, 2022-2023

v3.2 (07.09.22)

This booklet has a list of programming exercises sorted by (more or less) increasing difficulty. The student is advised to try all of them if possible, given that exercises for the lab exams will be very similar to the ones present here. In each one of them, the program must meet the requirements given. For omitted details, any reasonable decision on the part of the student will be accepted.

In all exercises, and unless the exercise tells otherwise, use standard input and output to get data and display results, respectively.

BLOCK 1.

1. Create a program which outputs a table of consecutive numbers with their respective squares. The program shall request from the user the value of the first number in the series and how many numbers to process.

2. Design a program to compute the average value of a list of numbers introduced by the user. The user will specify first the total number of elements in the list (that will be lower than a constant fixed in the program).

3. Sort the integer numbers in a list introduced by the user into an 'even' and an 'odd' list. The output of the program will be two lists, one with the even numbers in the original list, and another one with the odd ones. All three lists shall have a maximum size specified by constants in the program. The program must also ensure that the number of data read does not exceed that size.

4. Reverse the order of the characters in a text string (that is, show the characters in the list in reverse order). The original string and the inverted string will be displayed on the screen. The strings shall have a maximum length specified by a constant. The program must ensure that the characters in the string do not exceed that size.

5. Write the Fibonacci sequence. The program must read two integers from the keyboard. The first one will be the total number of elements to show in the sequence and the second one will be the first element to show. The Fibonacci sequence is:
    1. $n_0=0$;
    2. $n_1=1$;
    i. $n_i=n_{i-1}+n_{i-2}$;

6. Create a program that gets from the user an arbitrary text string consisting of letters and spaces, with less than 256 characters in total. The program must reduce to just one space the distance between two letters or words and show the resulting string in the screen.

7. Create a program that obtains a character string and a character from the user and displays the same string on the screen but without the specified character.

8. Create a program that obtains a string of characters from the user (the maximum size shall be defined by a constant in the program) and prints on the screen the uppercase letters present in that string.

9. Write a program that calculates the sum and subtraction of two amounts of money expressed in the Victorian English system: pennies, shillings and pounds. Note that 1 shilling = 12 pennies, and 1 pound = 20 shillings.

Further exercises for Block 1

1. Generate a table of equivalences between kilometers and nautical miles. The program will botain the interval from the user (e.g.: from 10 to 100 kilometers) and the step (e.g.: 10 kilometers), and will display a table where each line shows the distance in miles which corresponds to each distance in kilometers (e.g.: 10 km is 18.52 nautical miles, 20 km is 37.04 nautical miles, and so on). Take a nautical mile to be 1.852 kilometers.

2. Calculate the scalar product of two vectors. The program will ask the user to introduce the dimension of the vectors (less than a given constant) and the elements of each vector.

3. Create a program which obtains a string of characters from the user (with a program-specified maximum size) and two individual characters. The program will then show on the screen the original string, but inserting the second character after each occurrence of the first one.

4. Make a calculator to add and subtract hours expressed in HH:MM:SS format. The program will ask the user for two times and will return their sum and difference expressed in that same format. For example:

```
Introduce the first time   10:29:12

Introduce the second time:  8:42:25

Addition:                  19:11:37

Subtraction:                1:46:47
```

5. Design a program that changes lower case letters for upper case and vice versa. The program will read a character string from the keyboard and will show the result on the screen.

6. Design a program that asks the user for two character strings and shows them mingled, i.e., alternately taking a character from each one. For example, if the strings are: “elephant” and “CAT”, the program should display: “eClAeTphant”

7. Check if a character string is a palindrome. A palindrome is a string of text that reads the same forward as backwards. Whitespace characters don’t count for this.

8. Create a program that obtains from the user two vectors of arbitrary length (but equal for both and less than 20) and calculates the sum of both vectors, but previously rotating the first one a number of positions to the right, as indicated by the user. Rotation means that element `i` moves to position `(i+s) mod l`, where `s` is the step specified by the user and `l` the length of the vector.

9. The program will obtain from the user a list of numbers (less than 100), and from each non-overlapping group of three consecutive numbers, it will exchange the positions of the first and third numbers, showing the result in the screen. If the last group is not complete, it will be padded with zeros.

10. Create a program that reads a list of numbers from the keyboard with a length specified by the user. The program will display another list on the screen where the item in position *i* is the average of the items of positions 0 to *i* in the original list.

11. Add a certain number of elements of an arithmetic progression. The first element of the progression, the ratio, and the number of elements to be added, will be specified by the user. An arithmetic progression has the form

$$a_{i+1}=a_i+r$$

12. Create a program that obtains two positive integers from the user and indicates if they are mutually prime.

    Note: Two numbers are mutually prime if they do not share any dividers, i. e. if their maximum common divider is 1. For example, 12 and 5 are, while 10 and 6 are not.

13. Euclid realized that the maximum common divisor (*mcd*) of two numbers `a` and `b` (where `a>b`) is the same as the *mcd* of `b` and `r`, where `r` is the remainder of the division of `a` over `b`. If we now take `b` and `r` in place of `a` and `b`, we can repeat this process as long as we need, until the remainder obtained is finally `0`. At this moment, `b` is exactly the *mcd*. Use this idea to calculate the *mcd* of two arbitrary (positive) numbers.

14. Calculate the summatory of $1/n^2$ from n=1 until a maximum value specified by the user. The program will ask the user for the maximum value of *n*, and it will provide as output the total sum of values $1/1^2 + 1/2^2 + 1/3^2 + \ldots + 1/n^2$.
    Note: math tells us that, given that *n* is sufficiently high, this value will be close to $\pi^2/6$ (approximately 1.64493). A word of caution: division must be in floating point so as not to lose precision.

15. Create a program that sums the elements of Padovan's succession between two positions indicated by the user (both included). Padovan's succession is defined as:

    $p_0=p_1=p_2=1;$

    $p_n=p_{n-2}+p_{n-3};$

BLOCK 2.

1. Evaluate a polynomial of arbitrary degree at point x_0. Both the degree of the polynomial and point x_0 shall be specified by the user.

2. Compute the sum of two matrices. The matrices shall have maximum allowed dimensions specified by constants. The program will ask the user for the actual dimensions of the matrices, and will verify that they do not exceed the maximum limits. The result matrix will be displayed by rows.

3. Extract the main diagonal from a matrix. Show the original matrix row by row and the main diagonal in a vector. The maximum dimensions of the matrix shall be fixed by constants. The program must ensure that the number of elements does not exceed those constants.

4. Eliminate the consonants from a text string introduced by the user and show both strings on screen. Strings shall have a maximum length specified by a constant and the program must ensure that the characters in the string do not exceed it.

5. Create a program that obtains a character string from the user and substitutes the most frequent vowel in it with the least frequent one (if there are several that are the least frequent, one of them will be chosen arbitrarily).

6. When a matrix is big, but many of its elements are 0, it can be stored using the compress format. In this format, only a list of those elements which are different from 0, plus their rows and columns indices, are stored. Example (remember that in C, indexes start at 0):

   Original matrix:

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix}$$

   Compressed matrix: (1,0,1), (4,1,3), (2,3,0)

   Convert a sparse matrix into a compressed format. The program shall obtain from the user the size of the matrix and its elements. Then, it shall show both matrices. The original matrix will be displayed row by row. The compressed matrix is a single triplet list (value, row, column). The maximum dimensions of the matrix and the triplet list shall be fixed by constants. The program must ensure that the numbers of elements do not exceed them.

7. The program must ask the user to introduce a matrix and a vector and compute the vector obtained by multiplying the matrix by the vector. The maximum dimensions of the matrix and the vector shall be fixed by constants and the program must ensure that the number of elements does not exceed them.

8. Show a list of the prime numbers between two given numbers.

9. Construct a histogram of the vowels in a text. A text will be read from the keyboard, and the number of times each vowel appears in it (its histogram) will be displayed.

10. Count the number of words in a text. A text will be read from keyboard and the program will show how many words are in it. Consider that the text contains only letters, spaces and punctuation marks (dot, comma, colon, semicolon). After a punctuation mark there may or may not be a blank space.

11. Make a histogram of the results of the "roll a die" event. A histogram is a list that counts, for each possible result, the number of times that result has occurred. For example, the histogram: 5, 3, 7, 2, 6, 6, 1 would indicate that 1 has occurred 5 times, 2 3 times, and so on. To simulate the rolling of a die, use `srand()` and `rand()` functions in the `stdlib.h` library (See note (*) at the end of this document).

12. Repeat exercise 11, but rolling two dice

13. Draw on screen a pyramid built with the character '`*`'. The program will ask the user for the number of floors of the pyramid and show it centered on its base. Example:

```
4 floors:    *
            ***
           *****
          *******
```

14. Display an integer as ASCII encoded binary (See note (**) at the end of this document). Both the integer and the result of its conversion will be displayed.

Further exercises for Block 2.

15. Extract the secondary diagonal from a matrix. Show the original matrix row by row and the secondary diagonal as a vector. The maximum dimensions of the matrix shall be fixed by constants in the program. The program must ensure that the number of elements does not exceed those constants.

16. Print the first character of each word contained in a character string. The program shall obtain a string of characters from the user. Then, it shall first make sure that in contains only letters, whitespaces and punctuation signs (only comma, semi-colon, colon and period are allowed), otherwise it shall terminate, notifying the error. If it is a valid string, the program shall print in the next line the first character in every word, leaving a space between them.

17. Create a program that obtains from the user a matrix of arbitrary dimensions (but less than 20, and not necessarily square), and tells which is the column, the sum of whose elements is the biggest, and the value of that sum.

18. Calculate the solution of a second-degree equation. Note that the equation may have real **or complex** values as a solution.

19. Create a program that reads from the keyboard a matrix of user-specified dimensions (less than 20). The program will generate from that matrix another matrix with one more row and one more column. In each element of the new column, it will store the average of the rest of the elements in the same row. In each element of the new row, it will store the average of the other elements of its same column. In the element that belongs to both the new row and the new column, it will store the average of all the elements. The data will be floating point numbers.

20. Print the list of all pairs of integers lying between two user-defined limits whose sum is equal to a given integer number (without repetition, i.e., pair (a,b) is the same as pair (b,a)). Pair (a, a) is valid.

21. Create a program that obtains from the user a square matrix (with dimension lower than 20) and shows the transpose over the secondary diagonal.

22. Repeat exercise 13 showing only the edges of the pyramid.

23. Repeat exercise 13 showing an inverted pyramid.

24. Repeat exercise 13 showing a diamond.

25. Repeat exercise 13 showing a diabolo. Example for 5 floors:

```
*****
 ***
  *
 ***
*****
```

26. Implement the Newton-Raphson method to obtain the square root of a number a. This method consists of iterating the following formula:

$$x_{n+1} = \frac{x_n^2 + a}{2x_n},$$

with x_0=1. The higher the value of n, the closer x_n comes to the square root of a. The program will get the maximum root error E from the user. Then, iterations will stop when

$$|x - x_n| < E$$

27. Count the number of times a particular pattern appears in a line of text. A pattern will be read first, followed by the text, and the program will show the number of occurrences (0 occurrences is acceptable). Both the line and the pattern shall have maximum lengths fixed by a constant in the program. The program must make sure that the number of characters read does not exceed the available space.

28. Create a program that obtains from the user an arbitrary long text string (but less than 256 characters) and a positive number. It will then show the string rotated as many positions to the right as the number indicates. It will repeat the rotation as many times as necessary until the string becomes the original one again.

Example:

```
Type in the string: Hello world!
Rotation: 3
    'Hello world!'
    'lo world!Hel'
    'world!Hello '
    'ld!Hello wor'
    'Hello world!'
```

29. The program will obtain from the user a square matrix with a dimension less than 10, and a positive number. It will then display the result of rotating the matrix columns to the right as many positions as the number indicates.

    Example:

```
Matrix:       Number: 1           Rotated Matrix:
1 3 2 5                           5 1 3 2
2 2 3 0                           0 2 2 3
7 4 2 1                           1 7 4 2
```

30. Create a program that obtains from the user a matrix of integers (with less than 10 rows and 10 columns), and a positive integer. The program will display the result of removing from the matrix the row and column corresponding to the integer obtained from the user.

    Example:

```
Matrix:     Number:2          Output matrix:
1 3 2 5                       1 3 5
2 2 3 0                       2 2 0
7 2 4 1                       4 1 0
 4 1 4 0
```

31. Create a program that changes a number in base 10 to another user-eligible base. The letters of the alphabet will be used to represent digits greater than 9, starting at A for 10 up to Z for 23. It will ask for the new base first, then the number, and then it will print the result. Note that the maximum possible value for the base is 24.

32. Create a program that calculates the time interval between two dates in the same year, both included. Conditions:
    -Dates are expressed by a pair of numbers, which define day and month. For example, (3,11) would be November 3rd.
    -The time interval is expressed by a pair of numbers defining the number of weeks and the number of days, where the latter may not exceed 6 (as this would indicate one more week). For example: (5,3) would be 5 weeks + 3 days = 38 days.

    Example:

```
Date 1: 16,8    Date 2: 10,9

Result: 3 weeks and 5 days.
```

    Explanation: Between August, 16th and September, 10th there are 16 days in August and 10 in September (both dates are included). In total, 26 days, which makes 3 weeks and 5 days.

33. Create a program that requests the user to introduce a matrix of floating point numbers (with maximum dimension 10) and a vector with a maximum of 10 elements. The program will indicate whether the matrix contains the vector in any of its columns.

BLOCK 3.

1. Write a program that reports the rate of occurrence for each letter with respect to the total letters introduced by the user. The program will read from the keyboard an arbitrarily long character string ending with the end of file character, EOF (in Linux, Control-D).

2. Compute the multiplication of two matrices. The matrices shall have maximum dimensions specified by constants. The program will first ask the user for the actual dimensions of the matrices, and will verify that the multiplication is possible.

3. Implement a basic calculator (sum, subtraction, multiplication and division) that allows working in binary, octal, hexadecimal and decimal. The user will specify the representation system, the numbers, and the operation to perform with them. The result will be displayed in the same representation system. Divisions by zero must be avoided. The program will continuously ask for a new operation until the user presses a specific key to exit.
4. Compute the combinatorial number $\binom{m}{n}$ con $0 \leq n \leq m$.

   Note that $\binom{m}{n} = \binom{m-1}{n-1} + \binom{m-1}{n}$ and that $\binom{m}{0} = \binom{m}{m} = 1$. Thus, recursion is suggested.

5. Replace one pattern with another in a string of characters. The program will read a text, then the pattern to be replaced, and finally the pattern to replace it with. It will then show the modified string. If the pattern has not been found, the original string will be displayed.

Further exercises for Block 3.

6. Generate the parity bit of an integer. The parity bit is a bit that is added to an integer so that the total number of bits in the integer with value 1 (parity bit included) is even (for even parity bit) or odd (for odd parity bit). For this exercise, use even parity.

7. Sort a list of numbers. The program will obtain from the user a list of numbers and then will display it sorted in ascending order. You can use the *'bubble algorithm*' (See note (***) at the end of this document).

8. Sort a list of words. The program will get the list from the user and then the list will be displayed in alphabetical order.

9. Create a filter program: the program will read a pattern and then a multi-line text. It will display only those lines of text that contain the pattern.

10. Given an integer, calculate the number obtained by inverting the order of its bits. The program will read the number, and then display the bits of the original number and those of the result number, both in ASCII encoded binary (See note (**) at the end of this document).

11. The program will obtain from the user two lists of numbers and will select the number that, transferred to the other list, makes the sums in both lists as similar as possible. The lists are not necessarily the same length. The program will indicate which number of the list must be moved (it is not necessary to show the modified lists).

    Suggestion: From the list with the largest sum, choose the number whose double is closest to the difference in the sums

Example:

List1: `2, 5, 1, 7, 3`

List2: `5, 3, 6, 8, 1`

The sum of list1 is 18, and of list2 is 23. The difference is 5. The number on list2 whose double is closest to 5 is 3. If number 3 is moved to list1, the sum for this list will be 21, and for list2 it will be 20. Thus, the difference between the two lists is as small as possible.

12. Create a program that solves basic operations of numbers expressed in "reverse Polish" notation. The program will read the sequence of operations in the form of a single string of characters of arbitrary length and display the result on screen. In the string, operators and operands shall be separated by a single blank space. You can assume that the string read is correct. The null string is permissible.

    Example:
    String introduced: 5 2 + 3 *
    Result: 21

    Explanation:
    The first two numbers (5 and 2) are the operands of the next operation (+). 5+2=7. The result (7) is the first operand of the next operation (multiplication) and 3 is the second. 7*3=21.

13. Make a program that reads a sentence and, keeping the order of the words, writes each one backwards. For example:

    Input: `This is an easy program`

    Output: `sihT si na ysae margorp`

BLOCK 4.

1. Determine if one matrix is contained in another. The program will obtain two matrices from the user (with dimensions specified by them) and will indicate if one is contained in the other.

2. Make a basic "translator". The program will load from a file a series of correspondences between words. Then, for each word read from the keyboard, it will show its translation (if it exists; otherwise, it will leave it the same). The process will be repeated until a predefined keyword is read.

3. Solve a word search. The program will load a character matrix from a file. It will then read a word and search for it in the matrix in any orientation (vertical, horizontal, diagonal). If it finds it, it will indicate its location by the coordinates of its start letter and its end letter. If it does not find it, it will indicate it as well.

4. Manage a simple database. The database elements will have a code field (integer), a name field (string of characters), and an age field (integer). The only operations supported are: insert new element, delete element, and display element. *Insert* creates a new element, and requests its fields. *Delete* requests a code, and if it is present in the list, deletes it. *Show* requests a code and displays the information associated with the corresponding element. Assume that the code for each item is unique.

5. Repeat exercise 1 of block 3 with matrices in sparse format.

6. Calculate the determinant of a square matrix of arbitrary size. Given a square A matrix of size n×n, the sub-matrix $A_{p,q}$ is defined as the matrix of size (n - 1) × (n - 1), which results from eliminating in A row p and column q. With this, the following recursive formula allows us to calculate the determinant of A:

$$de(A) = \sum_{j=1}^{n} (-1)^{j+k} \cdot a_{kj} \cdot \det\left(A_{k,j}\right),$$

where a_{kj} is the element of row k and column j in A. In words, the determinant of A is calculated as a function of the determinants of its submatrices. The value of row k is arbitrarily fixed from among the values that meet that

$$1 \leq k \leq n.$$

Recursion is terminated when a sub-matrix has a single row and a single column, in which case it has a single element, which coincides with its determinant.

NOTES:

(*) The `random` and `srandom` functions of the `stdlib.h` library allow generating pseudo-random numbers. Each time it is invoked, `random` returns a pseudo-random integer between the values 0 and (2^31)-1. The numbers are obtained from a sequence generated by an algorithm that makes them look random. However, this is not really the case. Every time a program is executed, the sequence of numbers returned by the random function is the same. To avoid this, you should use the `srandom` function at the beginning of the program. This function allows you to set the "seed", that is, the position of the sequence by which the numbers start to be generated. It receives a parameter, which is a number from which it obtains the initial position of the sequence, so that, if each time the program is executed, the `srandom` function uses a different number as "seed", the numbers generated by the random function will be different. One way for the number used as seed to be different in each invocation of the program is to use the time function of the `time.h` library, which returns an integer with the time in seconds from 0 hours, 0 minutes and 0 seconds since January 1, 1970. Since this count is different every time the program is executed, this value can be used as seed for the `srandom` function.

Evidently, for the debugging of the program the interesting thing is that the sequence is precisely the same always, so, in this case, the value used as seed will always be the same.

(**) The ASCII encoded binary is a code that represents each bit of the data by an ASCII character, either '`0`' or '`1`', depending on the bit value. In this way, an integer is represented by a string of ASCII characters. For example, the decimal number 5 in pure 8-bit binary, would be represented as follows:

```
00000101.
```

Therefore, in ASCII encoded binary, an ASCII code would be used for each bit of the number:

```
'0' '0' '0' '0' '0' '1' '0' '1' '0',
```

where `'0'` stands for the ASCII code of 0 and `'1'` stands for the ASCII code of 1.

(***) The bubble algorithm, in the variant proposed for this exercise, consists of going through the list repositioning the elements that are in the wrong relative order. Consecutive elements of the list are compared, and if they are in the wrong order, their positions are switched. Then you move on to the next pair and repeat the operation. When the whole list has been run through, it is evaluated if any repositioning has been carried out. If this is the case, a complete run of the list is made again. Otherwise, it is already sorted. For example, to order the list in ascending order:

```
1, 4, 5, 3
```

take the first pair (1, 4). As it is in the right order, nothing is done. Then you move on to the next pair (4, 5). It is also in the correct order, so you move on to the next pair (5, 3). This is in the wrong order, so their positions are switched (3, 5). After scrolling through the entire list, the result is:

```
1, 4, 3, 5.
```

As one pair (3, 5) has been repositioned in the complete run, the process must be repeated again. Take the first pair (1, 4), leave it as it is, take the second (4, 3), which must be reordered to (3, 4), move on to the next ((4, 5), since 4 has taken the previous position of 3), and as it is ordered, leave it so. The list, after going through it, has become:

```
1, 3, 4, 5
```

Given that a modification has been made in the last run, it is necessary to make another one. Obviously, the list is already ordered, so nothing will be modified in this run, and that way you get to the end of the process.