CS19003 Programming and Data Structures Laboratory
Assignment 5

- ▷ Name the files as ⟨assignment no⟩_⟨question no⟩_⟨roll no⟩.c, without the ⟨ and ⟩. Consult your mentor for any confusion. Penalty if the file names do not stick to this convention.
- ▷ This is an assignment on one-dimensional arrays, functions and recursion. Do not use two-dimensional/multi-dimensional arrays, pointers, structures, dynamic memory allocation or any other advanced concept.
- ▷ If you need to declare arrays of variable size, you may use the method suggested in the last assignment.

---

1. Write a program that
   - ▷ asks the user to enter two non-negative integers as inputs. The input integers can be arbitrarily large. In particular, they may not fit in any integer data type (for example, they can have 100 digits).
   - ▷ stores the input integers in two strings (note: not integer arrays or variables of any integer type; also refer to point 1) using *scanf("%s", stringname)*.
   - ▷ computes and prints their difference to the screen. You may use an extra array for this step.

   You may use **only the string library functions that are present in your PDS theory slides**.

   Warning: Do not try to compute the values of the two numbers, store them in integer variables, and then use the C subtraction operation. As mentioned before, the numbers may be very large; their values may not fit in any integer data type. For a hint, you may remember how you do subtraction by hand starting from the unit's place.

   Sample input output (here the numbers are all small; however, we will test your programs on really large numbers):

   1) Enter the number of digits of a and b: 2 3

   Enter a: 23

   Enter b: 131

   Difference: -108

   2) Enter the number of digits of a and b: 4 4

   Enter a: 5651

   Enter b: 2345

   Difference: 3306

[100 points]

2. Write a **recursive** function named *nextnum* with appropriate prototype that takes a decimal number as an integer array of non-zero digits as input, and compute the next smallest number that is formed of the same digits (including repetitons). Eg. If the input number is 1245446, the function must compute 1245464. If the entered number is the largest number formed of those digits (eg. 544), the function should report that. Write a program that

   (a) takes in an integer n as input,

   (b) takes in n non-zero digits (i.e., from $1, \ldots, 9$) as inputs and stores them in an integer array,

   (c) call *nextnum* from the main and prints the next smallest number formed of the digits of the entered number, or reports that the input number is the largest number formed of its digits.

   Note: A perfect non-recursive program will fetch much less marks than a partly correct recursive program. So, please use recursion in your program. For a hint, note that finding the next largest number of 999923 involves (recursively?) finding the next largest number of 23.

   Sample input/output:

   How many digits (n)? 20

   Enter digit 1: 3

   Enter digit 2: 4

   Enter digit 3: 1

   Enter digit 4: 6

   Enter digit 5: 9

   Enter digit 6: 4

   Enter digit 7: 2

   Enter digit 8: 2

   Enter digit 9: 2

   Enter digit 10: 6

   Enter digit 11: 5

   Enter digit 12: 1

   Enter digit 13: 9

   Enter digit 14: 8

   Enter digit 15: 7

   Enter digit 16: 6

   Enter digit 17: 5

   Enter digit 18: 4

   Enter digit 19: 3

   Enter digit 20: 2

   Input number: 34169422265198765432

   Next smallest number: 34169422265213456789

How many digits (n)? 5

Enter digit 1: 9

Enter digit 2: 8

Enter digit 3: 7

Enter digit 4: 6

Enter digit 5: 5

Input number: 98765

Number maximum

[100 points]

3. Write a program **using recursion** that

   (a) takes a decimal number A as a string as input from the user, using *scanf("%s", stringname)*. You may assume that the number of digits is at most 200.

   (b) takes a non-zero digit as input from the user and stores it in an integer variable n,

   (c) divides the number stored in A with the digit n, and prints the quotient and the remainder. This step must be done recursively. Functions with appropriate prototypes may be defined. You may use **only the string library functions that are present in your PDS theory slides**.

   Note: A perfect non-recursive program will fetch much less marks than a partly correct recursive program. So, please use recursion in your program. As a hint, imagine performing the division manually. After the first step, are you not left with the task of divising a smaller dividend by the same divisor?

   Sample input/output:

   Enter number: 76654323754

   Enter divisor (digit): 7

   Quotient: 10950617679

   Remainder: 1

   [100 points]