**INDIAN STATISTICAL INSTITUTE**

MTech(CS) I year 2022-2023

Subject: Computing Laboratory

Lab Test 1 (August 31, 2022)

Total: $4 \times 15 = 60$ marks Duration: 4 hours

**SUBMISSION INSTRUCTIONS**

1. All programs should take the required input via the standard input (terminal/keyboard), and print the desired output to the terminal.
2. Please make sure that your programs adhere strictly to the specified input and output format. Do not print extra strings asking the user for input, debugging messages, etc. These will cause the automatic checking system to fail.

Q1. Write a C program that takes in an array of 0s and 1s, and rearranges the array so that all 0s are at the beginning and all 1s are at the end.

**Input format:** You will be given a sequence of 0s and 1s. The number of 0s and 1s will not be known to you in advance (but it will be small enough to be stored in a variable of `int` type).

**Output format:** Your program should print the rearranged array.

**Sample input 1**

```
0101010101
```

**Sample output 1**

```
0000011111
```

**Sample input 2**

```
00000
```

**Sample output 2**

```
00000
```

**Sample input 3**

```
11111
```

**Sample output 3**

```
11111
```

Q2. Write a C program to compute the product of two positive integers. The input integers may contain up to 100 digits (0–9). Thus, you will not be able to use the usual builtin types (`int`, `long int`, etc.) to store the inputs. You may **not** use the GNU Multiple Precision (MP) Arithmetic Library, or any other similar library or C++ class.

**Input format:** You will be given the 2 positive integers as strings via standard input (i.e., the terminal). Each integer will consist of a sequence of up to 100 digits. The strings will be separated by one newline character.

**Output format:** Your program should print the product of the two numbers. For full credit, the result should not contain any leading zeros.

**Sample input 1**

```
92233720368547758078771
922337203685477580798221876
```

**Sample output 1**

```
850705917302346158645460964428159973034107433944 96
```

**Sample input 2**

```
10000000000000000000000000000000000000000050
10000000000000000000000000000000000000000050
```

**Sample output 2**

```
100000000000000000000000000000000000000001000000000000000000000000000000000000002500
```

Q3. *Forward Backward Sort* is a sorting algorithm that can be used to arrange the elements of an array in increasing order. The essential idea here is that in each iteration, we traverse the array *twice*: first from left to right, and then from right to left. For your convenience, we provide the algorithm below. Each iteration of the algorithm is broken up into 2 stages:

- The first stage loops through the array from left to right. During the loop, adjacent items are compared; if the value on the left is greater than the value on the right, then the values are swapped. At the end of the first stage, the largest number will reside at the end of the array.
- The second stage loops through the array in the opposite direction, starting from the last but one item, and moving back to the start of the array. Here also, adjacent items are compared and are swapped if required, so that at the end of the second stage, the *smallest number* will be moved to the first position of the array.

The second iteration is similar to the first, except that the two elements at the ends of the array (the minimum and the maximum) are no longer considered. Thus, the second iteration works with $n - 2$ elements (where $n$ is the number of elements in the input). Similarly, the third iteration works with $n - 4$ elements.

If, during a pass over the array, no swap occurs, the algorithm stops. At this point, all elements in the array are arranged in increasing order.

**Example:** Let us consider an example array (5 1 4 2 8 0 2)

First Forward Pass:
(5 1 4 2 8 0 2) $\implies$ (1 5 4 2 8 0 2), Swap since $5 > 1$
(1 5 4 2 8 0 2) $\implies$ (1 4 5 2 8 0 2), Swap since $5 > 4$
(1 4 5 2 8 0 2) $\implies$ (1 4 2 5 8 0 2), Swap since $5 > 2$
(1 4 2 5 8 0 2) $\implies$ (1 4 2 5 8 0 2), No swap since $5 \leq 8$
(1 4 2 5 8 0 2) $\implies$ (1 4 2 5 0 8 2), Swap since $8 > 0$
(1 4 2 5 0 8 2) $\implies$ (1 4 2 5 0 2 8), Swap since $8 > 2$
After the first forward pass, the greatest element (8) moves to the last position in the array. Next, the first backward pass starts.

First Backward Pass:
(1 4 2 5 0 2 8) $\implies$ (1 4 2 5 0 2 8), No swap since $0 \leq 2$
(1 4 2 5 0 2 8) $\implies$ (1 4 2 0 5 2 8), Swap since $5 > 0$
(1 4 2 0 5 2 8) $\implies$ (1 4 0 2 5 2 8), Swap since $2 > 0$
(1 4 0 2 5 2 8) $\implies$ (1 0 4 2 5 2 8), Swap since $4 > 0$
(1 0 4 2 5 2 8) $\implies$ (0 1 4 2 5 2 8), Swap since $1 > 0$
After the first backward pass, the smallest element (0) moves to the first position of the array.

Second Forward Pass:
(0 1 4 2 5 2 8) $\implies$ (0 1 4 2 5 2 8), No swap since $1 \leq 4$
(0 1 4 2 5 2 8) $\implies$ (0 1 2 4 5 2 8), Swap since $4 > 2$
(0 1 2 4 5 2 8) $\implies$ (0 1 2 4 5 2 8), No swap since $4 \leq 5$
(0 1 2 4 5 2 8) $\implies$ (0 1 2 4 2 5 8), Swap since $5 > 2$

Second Backward Pass:
(0 1 2 4 2 5 8) $\implies$ (0 1 2 2 4 5 8), Swap since $4 > 2$

Now, the array is already sorted, but our algorithm doesn't know this. The algorithm completes a third forward pass, during which no swap occurs. It then stops.

(0 1 2 2 4 5 8) $\implies$ (0 1 2 2 4 5 8)

**Input format:** You will be given $n \leq 100$ integers as inputs. The integers will be separated by spaces. The number $n$ itself will **not** be provided as input.

**Output format:** Your program should print the $n$ input integers in ascending order in one single line. The integers should be separated by one blank space. You will not get credit if your program implements a sorting algorithm different from the one described above.

**Sample input 1**

```
0 -9 -10 204 81 999 -9883 81 1000
```

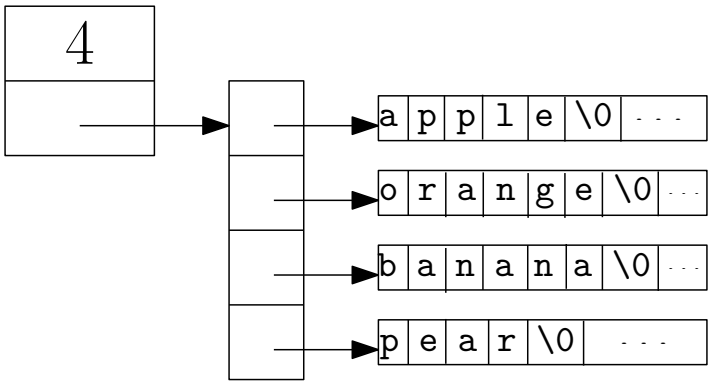
**Sample output 1**

```
-9883 -10 -9 0 81 81 204 999 1000
```

Q4. Implement a list of strings using dynamic memory. The list consists of a header storing two items:

- the size of the list, and
- a dynamically allocated array capable of storing only the elements in the list.

In the picture below, the list consists of the strings `apple`, `orange`, `banana`, `pear`. Its size is 4, and the dynamic array is large enough to store exactly four strings.




Define a structure to store the size of the list, and a dynamically allocated array that stores the elements (strings) of the list. You may assume that each element (string) will contain **at most 9 lowercase letters.** Use a `typedef` to define `LIST` as the name of your structure.

Implement the following functions.

(a) `create_list(void)`: returns an empty list. For an empty list, the size is zero, and the array is NULL.

(b) `print_list(LIST L)`: print the elements of the list separated by spaces, and terminated by a newline.

(c) `append(LIST L, char *a)`: appends the string `a` to the end of the list `L`, and returns the modified list.

(d) `prepend(LIST L, char *a)`: prepends the string `a` at the beginning of the list `L`, and returns the modified list.

(e) `deletelast(LIST L)`: deletes the last string of the list, and returns the modified list.

(f) `deletefirst(LIST L)`: deletes the first string of the list, and returns the modified list.

(g) `deleteall(LIST L, char *a)`: deletes all occurrences of the string `a` in `L`, and returns the modified list. Note that the string `a` must occur as an element in `L`. If `a` is a non-trivial substring of an element, then that element must not be removed. For example, if `a` is the string `pea`, and `pear` is an element of `L`, the element `pear` should not be removed.

Do not forget to use `free()` where necessary.

Test your code using the following sequence of operations. Use `print_list` to display the list after each operation.

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| create_list() | append(banana) | prepend(orange) | deletefirst() |
| append(apple) | append(pineapple) | prepend(apple) | deletefirst() |
| append(lemon) | prepend(apple) | prepend(lemon) | deletefirst() |
| append(banana) | prepend(banana) | deletelast() | deletefirst() |
| append(pineapple) | prepend(cherry) | deletelast() | deleteall(plum) |
| append(pear) | prepend(orange) | deletelast() | deleteall(banana) |
| append(mango) | prepend(banana) | deletelast() | deleteall(apple) |
| append(orange) | prepend(plum) | deletelast() | deleteall(cherry) |
| append(cherry) | prepend(apple) | deletefirst() | deleteall(mango) |

### Sample output

```
()
(apple)
(apple,lemon)
(apple,lemon,banana)
(apple,lemon,banana,pineapple)
(apple,lemon,banana,pineapple,pear)
(apple,lemon,banana,pineapple,pear,mango)
(apple,lemon,banana,pineapple,pear,mango,orange)
(apple,lemon,banana,pineapple,pear,mango,orange,cherry)
(apple,lemon,banana,pineapple,pear,mango,orange,cherry,banana)
(apple,lemon,banana,pineapple,pear,mango,orange,cherry,banana,pineapple)
(apple,apple,lemon,banana,pineapple,pear,mango,orange,cherry,banana,pineapple)
(banana,apple,apple,lemon,banana,pineapple,pear,mango,orange,cherry,banana,pineapple)
(cherry,banana,apple,apple,lemon,banana,pineapple,pear,mango,orange,cherry,banana,pineapple)
(orange,cherry,banana,apple,apple,lemon,banana,pineapple,pear,mango,orange,cherry,banana,pineapple)
(banana,orange,cherry,banana,apple,apple,lemon,banana,pineapple,pear,mango,orange,cherry,banana,pineapple)
(plum,banana,orange,cherry,banana,apple,apple,lemon,banana,pineapple,pear,mango,orange,cherry,banana,pineapple)
(apple,plum,banana,orange,cherry,banana,apple,apple,lemon,banana,pineapple,pear,mango,orange,cherry,banana,pineapple)
(orange,apple,plum,banana,orange,cherry,banana,apple,apple,lemon,banana,pineapple,pear,mango,orange,cherry,banana,pineapple)
(apple,orange,apple,plum,banana,orange,cherry,banana,apple,apple,lemon,banana,pineapple,pear,mango,orange,cherry,banana,pineapp
(lemon,apple,orange,apple,plum,banana,orange,cherry,banana,apple,apple,lemon,banana,pineapple,pear,mango,orange,cherry,banana,p
(lemon,apple,orange,apple,plum,banana,orange,cherry,banana,apple,apple,lemon,banana,pineapple,pear,mango,orange,cherry,banana)
(lemon,apple,orange,apple,plum,banana,orange,cherry,banana,apple,apple,lemon,banana,pineapple,pear,mango,orange,cherry)
(lemon,apple,orange,apple,plum,banana,orange,cherry,banana,apple,apple,lemon,banana,pineapple,pear,mango,orange)
(lemon,apple,orange,apple,plum,banana,orange,cherry,banana,apple,apple,lemon,banana,pineapple,pear,mango)
(lemon,apple,orange,apple,plum,banana,orange,cherry,banana,apple,apple,lemon,banana,pineapple,pear)
(apple,orange,apple,plum,banana,orange,cherry,banana,apple,apple,lemon,banana,pineapple,pear)
```

(orange,apple,plum,banana,orange,cherry,banana,apple,apple,lemon,banana,pineapple,pear)

(apple,plum,banana,orange,cherry,banana,apple,apple,lemon,banana,pineapple,pear)

(plum,banana,orange,cherry,banana,apple,apple,lemon,banana,pineapple,pear)

(banana,orange,cherry,banana,apple,apple,lemon,banana,pineapple,pear)

(banana,orange,cherry,banana,apple,apple,lemon,banana,pineapple,pear)

(orange,cherry,apple,apple,lemon,pineapple,pear)

(orange,cherry,lemon,pineapple,pear)

(orange,lemon,pineapple,pear)

(orange,lemon,pineapple,pear)