LAB 4 — Arrays and Pointers

Problem A

A.1 Specification

Write a C program to input a line of characters and store the input characters in an array. Reverse the order of the input characters and display the reversed string on the standard output using **`printf`**.

A.2 Implementation

- The program is named `lab4a.c`. **Use the given template `lab4a.c`** and fill in your code.
- You are given an array of characters of size `MAX_SIZE` where `MAX_SIZE` = 100. The array is named `my_strg`.
- Use `getchar` and a loop to read a line of characters, and store the input characters into the array. The loop terminates when a new line character '`\n`' is entered. The new line character '`\n`' is NOT part of the line (i.e., discard the new line character '`\n`').
- Reverse the order of the input characters stored in the array.
- Display on the standard output the reversed string using the `printf` statement as follows:

```
printf( "%s\n", my_strg );
```

- This is the same problem as lab3. The difference is that you are now given two pointer variables, p and v, that point to array `my_strg`. In your code, do not use variable `my_strg`. Use only p and v (and additional variables other than arrays, if needed) to manipulate the array elements. That is, you should never have to type the text "`my_strg`" when coding.

A.3 Sample Inputs/Outputs

```
indigo 352 % lab4a
Hello, world!
!dlrow ,olleH
indigo 353 % lab4a
Welcome to CSE2031.
.1302ESC ot emocleW
indigo 354 % lab4a
```

A

A

```
indigo 355 % lab4a
123
321
indigo 356 % lab4a
```

Problem B

B.1 Specification

Write a C program to input a line of characters in the form of a floating-point number, convert the line of characters into an actual floating-point number, and display on the standard output the floating-point number.

B.2 Implementation

- The program is named `lab4b.c`. **Use the given template `lab4b.c`** and fill in your code.
- You are given an array of characters of size `MAX_SIZE` where `MAX_SIZE` = 100. The array is named `my_strg`.
- Use `getchar` and a loop to read a line of characters, and store the input characters into array `my_strg`. The loop terminates when a new line character '`\n`' is entered. The new line character '`\n`' is NOT part of the line (i.e., discard the new line character '`\n`').
- The input line contains only characters '0' to '9' and the dot character '**.**' in the form of a valid positive floating point number of the following format: **[integer part] . [fractional part]**
- Convert the input line of characters to a **double** floating-point number which is stored in variable `my_number`.
- Display on the standard output the double floating-point number `my_number` using the `printf` statement as follows:

```
printf( "%.6f\n", my_number );
```

- Assume that the input line of characters represents a valid floating point number of the form [integer part] **.** [fractional part]

B.3 Sample Inputs/Outputs

```
indigo 360 % lab4b
24.5
24.500000
indigo 361 % lab4b
76.24
76.240000
indigo 362 % lab4b
100.0
100.000000
indigo 363 % lab4b
0.255
0.255000
indigo 364 % lab4b
12.9999999999
13.000000
indigo 365 % lab4b
1.00000000099
1.000000
indigo 366 % lab4b
```

Common Notes

All submitted files should contain the following header:

```
/**************************************
*     CSE2031 – Lab 4                 *
*     Filename:  Name of file         *
*     Author: Last name, first name   *
*     Email: Your email address       *
*     cse_num: Your cse number        *
**************************************/
```

In addition, all programs should follow the following guidelines:

- Include the `stdio.h` library in the header of your `.c` files.
- Use `printf` to print text and outputs according to the required formats.
- End each output result with a new line character '`\n`'.
- Do not use any C library functions except `getchar(), putchar(), scanf()` and `printf()`.
- **Assume that all inputs are valid (no error checking is required on inputs).**