ISLAMIC UNIVERSITY OF TECHNOLOGY (IUT)
ORGANISATION OF ISLAMIC COOPERATION (OIC)

Department of Computer Science and Engineering (CSE)

CSE 4108: Structured Programming I Lab
Lab 9, Section 2A

Objectives

- Strings

Guidelines

- All source codes (.c files) in the naming format `ID_Lab9_TaskN.c`, eg. `230041101_Lab9_Task1.c`.
- Ensure that your program produces the correct output for all sample cases.
- Source codes must be properly indented
- Screenshots are not required.
- Throughout the rest of the semester, try to regularly solve problems and participate in contests on online judges such as *Codeforces*, *Codechef*, *Atcoder*.

Useful String Functions in C

This lab will require usage of string functions. The function descriptions, syntax and usage examples of commonly used string functions are provided here for your convenience. Make sure to write `#include <string.h>` in your headers to use these functions.

1. `gets()`

- **Syntax:** `char *gets(char *str);`
- **Description:** Reads a line from `stdin` into the string `str` until a newline is encountered.
- **Example:**

```
char str[50];
printf("Enter a string: ");
gets(str);
printf("You entered: %s", str);
```

2. `strlen()`

- **Syntax:** `size_t strlen(const char *str);`
- **Description:** Returns the length of the string `str`, excluding the null terminator.

- **Example:**

```
char str[] = "Hello";
printf("Length of string: %d", strlen(str));
```

3. `strcpy()`

- **Syntax:** `char *strcpy(char *dest, const char *src);`
- **Description:** Copies the string `src` into `dest`.
- **Example:**

```
char src[] = "World";
char dest[50];
strcpy(dest, src);
printf("Copied string: %s", dest);
```

4. `strcat()`

- **Syntax:** `char *strcat(char *dest, const char *src);`
- **Description:** Appends the string `src` to the end of `dest`.
- **Example:**

```
char str1[50] = "Hello, ";
char str2[] = "World!";
strcat(str1, str2);
printf("Concatenated string: %s", str1);
```

5. Character Testing Functions (`islower()`, `isupper()`, `isalpha()`)

These functions from the `<ctype.h>` library are used to test individual characters. Remember to include `#include <ctype.h>` in your headers to use them.

- **`islower()`**:
  - **Syntax:** `int islower(int ch);`
  - **Description:** Returns a non-zero value if `ch` is a lowercase letter; otherwise, it returns 0.
- **`isupper()`**:
  - **Syntax:** `int isupper(int ch);`
  - **Description:** Returns a non-zero value if `ch` is an uppercase letter; otherwise, it returns 0.
- **`isalpha()`**:
  - **Syntax:** `int isalpha(int ch);`
  - **Description:** Returns a non-zero value if `ch` is an alphabetic letter (either uppercase or lowercase); otherwise, it returns 0.

Task 1 — Caesar Cipher Deciphering

In this task, you will write a program in C to decipher a string encoded using the Caesar Cipher. The Caesar Cipher shifts each letter of the string by a fixed number of positions in the alphabet. For this task, the shift value used for encoding is fixed at 3. Your program should reverse the cipher and retrieve the original string. For example, if a character is 'D', it would be deciphered (shifted back 3 positions) as 'A'.

The procedure to decipher the Caesar Cipher is as follows:

- For each letter in the string:
  - If the letter is uppercase (`A-Z`), shift it back by 3 positions in the alphabet. If the shift goes past `A`, it wraps around to the end of the alphabet.
  - If the letter is lowercase (`a-z`), shift it back by 3 positions in the alphabet. If the shift goes past `a`, it wraps around to the end of the alphabet.
- Non-alphabetic characters remain unchanged.

Input

The input consists of a single string of up to 100 characters. The string may contain spaces, mixed case letters, digits, and special characters.

Output

The output consists of the deciphered string after reversing the Caesar Cipher.

Sample Execution(s)

Input 1

```
Khoor, Zruog!
```

Output 1

```
Hello, World!
```

Input 2

```
Fdhvdu Flskhu
```

Output 2

```
Caesar Cipher
```

Input 3

```
Cd q Chh Lfh Fuhdp
```

Output 3

```
Za n Zee Ice Cream
```

Task 2 — String Compression

In this task, you will write a program in C to compress a string by replacing consecutive repeated characters with the character followed by the count of its occurrences. If the compressed string is not shorter than the original string, your program should output the original string instead.

The procedure to compress a string is as follows:

- Traverse the string and count consecutive occurrences of each character.
- Replace the sequence of repeated characters with the character followed by the count.
- Compare the length of the compressed string with the original string:
  - If the compressed string is shorter, print the compressed string.
  - Otherwise, print the original string.

Input

The input consists of a single string of up to 100 characters. The string contains only uppercase and lowercase letters.

Output

The output is the compressed version of the string if it is shorter than the original, or the original string otherwise.

Sample Execution(s)

Input 1

```
Enter the string: aabccccccaaa
```

Output 1

```
a2b1c5a3
```

Input 2

```
Enter the string: abcdef
```

Output 2

```
abcdef
```

Input 3

```
Enter the string: AaAaBbBb
```

Output 3

```
AaAaBbBb
```

Input 4

```
Enter the string: zzzzz
```

Output 4

```
z5
```

Task 3 — Substring Pattern Matching

In this task, you will write a program in C to find all occurrences of a given substring (pattern) within a larger string (text). The program should output the starting index of each occurrence of the substring in the text. The index starts at 0.

The program must handle both overlapping and non-overlapping matches.

Input

The input consists of two lines:

- The first line contains the text (a string of up to 100 characters).
- The second line contains the pattern (a string of up to 20 characters).

Output

The output consists of the starting indices of all occurrences of the pattern in the text, printed on a single line separated by spaces. If the pattern does not occur in the text, print `-1`.

Sample Execution(s)

Input 1

```
Enter the text: abcabcabc
Enter the pattern: abc
```

Output 1

```
0 3 6
```

Input 2

```
Enter the text: banana
Enter the pattern: ana
```

Output 2

```
1 3
```

Input 3

```
Enter the text: hello world
Enter the pattern: test
```

Output 3

```
-1
```

Input 4

```
Enter the text: aaa
Enter the pattern: aa
```

Output 4

```
0 1
```