HOMEWORK 1

Question 1 (50 points). Write a C program that reads a string (line) that user inputs in the terminal screen, and then prints the string with lowercases, uppercases, sentence cases and capitalized cases according to instructions below. The code template and a proper output example for this program are shown in Figure 1 and Figure 2 respectively. The extended ASCII table is given in Figure 3.

- The program should start to run by printing "Please enter a string " on the screen and then read the complete user input by using an appropriate <stdio.h> function at once. Then, then it should print the original, lowercased, uppercased, sentence cased and capitalized cased versions of the string.
- Function prototypes char* lowerCase(char *str); char* upperCase(char *str); char* sentenceCase(char *str); and char* capitalizedCase(char *str); should be implemented below int main() function.
- char* lowerCase(char *str) function should return the string as all letters of it are lowercased.
- char* upperCase(char *str) function should return the string as all letters of it are uppercased.
- char* sentenceCase(char *str) function should return the string as only the first letter of it is capitalized and rest of the letters are lowercased.
- char* capitalizedCase(char *str) function should return the string as the first letters of all words in the string are capitalized and rest of the letters are lowercased.
- The functions above should not print anything. All printing should be done inside the int main() function via print() function calls. The string should be printed with lowercases, uppercases, sentence cases and capitalized cases by calling char* lowerCase(char *str), char* upperCase(char *str), char* sentenceCase(char *str) and char* capitalizedCase(char *str) functions inside printf() functions respectively.

```c
#include <stdio.h>

char* lowerCase(char *str);
char* upperCase(char *str);
char* sentenceCase(char *str);
char* capitalizedCase(char* str);

int main(){
    // Define a char array of 100 characters.
    printf("Please enter a string: ");
    // Using an appropriate function from <stdio.h> library, read the string from the user input into the defined char array.
    // Using the "printf" function, print the string, which is read from the user input, on the terminal screen.
    // Print the string with lowercases, by calling your "lowerCase" function inside of the "printf" function.
    // Print the string with uppercases, by calling your "upperCase" function inside of the "printf" function.
    // Print the string with sentence cases, by calling your "sentenceCase" function inside of the "printf" function.
    // Print the string with capitalized cases, by calling your "capitalizedCase" function inside of the "printf" function.
    return 0;
}

char* lowerCase(char *str){
    // Implement the code that converts all letters in str to lowercase.
    return str;
}

char* upperCase(char *str){
    // Implement the code that converts all letters in str to uppercase.
    return str;
}

char* sentenceCase(char *str){
    // Implement the code that converts str to sentence case by capitalizing only the first letter of the string.
    // Hint: You may think to ease the implementation by using "lowerCase" function first.
    return str;
}

char* capitalizedCase(char* str){
    // Implement the code that converts str to capitilized case by capitalizing first letters of all words in the string.
    // Hint: You may think to ease the implementation by using "sentenceCase" function first.
    // Hint: You may identify words by detecting space characters.
    return str;
}
```

Figure 1. The code template for question 1

```
Please enter a string: hello World!

Original: hello World!

Lower case: hello world!

Upper case: HELLO WORLD!

Sentence case: Hello world!

Capitalized case: Hello World!
```

Figure 2. A proper output example for question 1

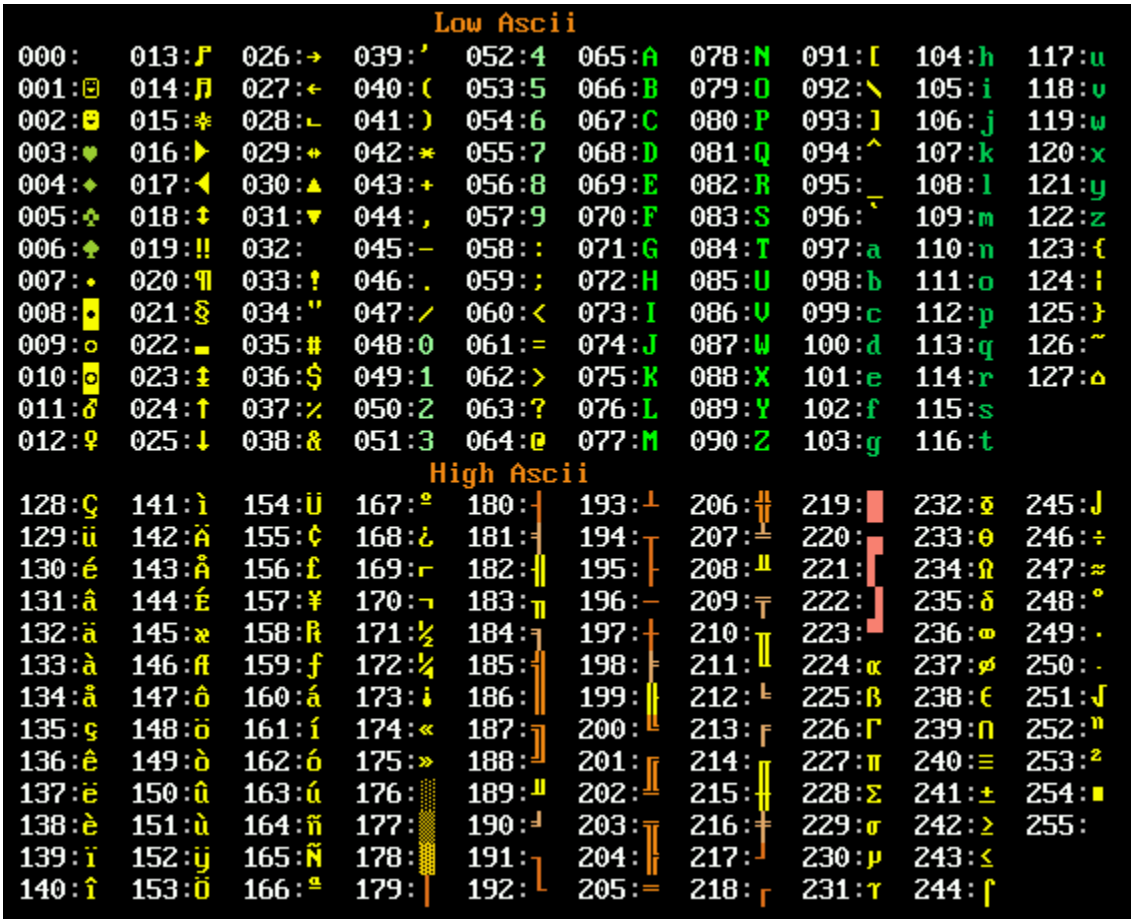

Figure 3. The extended ASCII table

Question 2 (50 points). Write a function with the prototype double ln(unsigned long long x, int n); that iteratively approximates to the natural logarithm of argument x using n iterations according to instructions below. The code template and a proper output example for this program are shown in Figure 4 and Figure 5 respectively.

- The program should start to run by creating a seed for rand() function by using the appropriate <stdlib.h> function that takes time() function from <time.h> as the argument. Then, it should define **x** as the multiplication of three random numbers generated by rand() function. Afterward, it should find **n** that is the minimum number of iterations required to reach an error below the error tolerance **t** = 1E-3. Error is the absolute value of the difference between the natural logarithms of **x** calculated by log() function from <math.h> and approximated by double ln(unsigned long long x, int n) function at n iterations. Finally, it should print the found minimum number of iteration **n** that satisfy the error condition, the generated **x** value, log(**x**) value, ln(**x**, **n**) value, and the error as the absolute difference between log(**x**) and ln(**x**, **n**) by using fabs() function from <math.h>.
- double ln(unsigned long long x, int n) function should check if logarithm is defined at argument x first; if not, it should print the message "Invalid argument!" via printf() function and terminate the program by calling <stdlib.h> function exit() immediately. If the argument x is valid, the function should iteratively approximate the natural logarithm of x at using n iterations. For the iterative approximation, Newton's method-based approximation which is given as per (1) should be implemented. To calculate exponential terms, <math.h> functions exp() or pow() may be used.

$$\ln x \approx y_n \text{ for } y_{i+1} = y_i + 2 \cdot \frac{x - e^{y_i}}{x + e^{y_i}} \text{ where } x \in \mathbb{R}^+, i \in \mathbb{N}, n \in \mathbb{Z}^+ \text{ and } i < n \quad (1)$$

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

double ln(unsigned long long x, int n);

int main(){
    unsigned long long x;   // value
    double t = 1E-3;        // error tolerance
    int n;                  // number of iterations required to reach an error below the error tolerance t

    // Create a seed for "rand" function based on "time" function
    // Define x as the multiplication of three random numbers generated via "rand" function
    // Find n

    printf("Iteration %d\nx\t= %llu\nlog(x)\t= %lf\nln(x)\t= %lf\nError\t= %lf\n", n, x, log(x), ln(x,n), fabs(log(x)-ln(x,n)));

    return 0;
}

double ln(unsigned long long x, int n){
    // Where the logarithm is undefined at x (x <= 0) print "Invalid argument!" with "printf" function
    // and terminate program using "exit" function.

    // Implement the Newton's method-based iterative approximation for natural logarithm of x using n iterations
    // Return the approximated natural logarithm of x
}
```

Figure 4. The code template for question 2

```
Iteration 24
x       = 18446744071893929672
log(x)  = 44.361420
ln(x)   = 44.361417
Error   = 0.000003
```

(a)

```
Invalid argument!

Process returned -1 (0xFFFFFFFF)
```

(b)

Figure 5. Proper output examples for question 2

IMPORTANT NOTES

1. The **due date** of this **Homework 1** is **20 NOVEMBER 2023** Monday at **23.55 sharp**.
2. This is a **group assignment**, and each group may consist of **2 members** at most.
3. Each group should submit **2 separate *.c files** for question 1 and question 2. Other file types will neither be accepted nor graded.
4. Only **one group member** should **submit** requested files on behalf of the group. Therefore, both group members' **full names** and **student numbers** must be specified on the top lines of each submitted *.c file as comments.
5. Submitted *.c files need to be **compiled** without any problem, otherwise the **grades** for those files will be cut.
6. Each functionality will be tested and graded.
7. Late submissions will reduce **20 points** per each day late.
8. **Cheating is strictly forbidden**! Once cheating is detected, all involved submissions will suffer the consequences (i.e., negated grades).