# **Project Report: C Programming Solutions**

## **Introduction**

This report presents three C programs developed as part of the **Programming in C** end-semester examination. The programs are:

1. **Readability Analyzer** (Coleman-Liau Index)
2. **DNA Complement Generator**
3. **Caesar Cipher Encryption**

Each program is designed to solve a specific problem using fundamental C programming concepts. This report explains the **logic, implementation, and key features** of each solution.

## **Program 1: Readability Analyzer (Coleman-Liau Index)**

### **Objective**

This program calculates the **U.S. school grade level** required to understand a given text using the **Coleman-Liau Index**.

### **Key Features**

* Counts **letters, words, and sentences** in the input text.
* Computes the **Coleman-Liau Index** using the formula: [ ]]\text{Index} = 0.0588 \times L - 0.296 \times S - 15.8 ] where:
  + ( L ) = average letters per 100 words
  + ( S ) = average sentences per 100 words
* Outputs the **grade level** (e.g., "Grade 5", "Before Grade 1", or "Grade 16+").

### **Implementation Steps**

1. **Input Handling** - The user enters a text string.
   * The program reads it using gets() (simplified input method).
2. **Counting Letters, Words, and Sentences** - **Letters**: Checks if a character is between A-Z or a-z.
   * **Words**: Increments count when a space ' ' is found.
   * **Sentences**: Counts ., !, and ? as sentence terminators.
3. **Calculating the Index** - Computes L and S (letters and sentences per 100 words).
   * Applies the formula and rounds the result.
4. **Output** - If the index is **< 1**, prints "Before Grade 1".
   * If the index is **≥ 16**, prints "Grade 16+".
   * Otherwise, prints the rounded grade.

### **Example**

**Input:** ``` Text: Hello, world! This is a test.

\*\*Output:\*\* ```  
Grade 3

## **Program 2: DNA Complement Generator**

### **Objective**

This program takes a **DNA sequence** and generates its **complementary strand** based on base-pairing rules:

* **A (Adenine) ↔ T (Thymine)**
* **C (Cytosine) ↔ G (Guanine)**

### **Key Features**

* Handles **uppercase (A, T, C, G)** and **lowercase (a, t, c, g)** input.
* Preserves the **original case** in the output.
* Prints the **complementary DNA sequence**.

### **Implementation Steps**

1. **Input Handling** - The user enters a DNA sequence (e.g., ATCG).
   * The program reads it using gets().
2. **Generating the Complement** - Loops through each character and replaces:
   * A → T (and a → t)
   * T → A (and t → a)
   * C → G (and c → g)
   * G → C (and g → c)
3. **Output** - Prints the complementary sequence.

### **Example**

**Input:** ``` Enter a DNA sequence: ATCG

\*\*Output:\*\* ```  
Complement: TAGC

## **Program 3: Caesar Cipher Encryption**

### **Objective**

This program **encrypts a message** using the **Caesar Cipher**, where each letter is shifted by a given key.

### **Key Features**

* **Preserves uppercase/lowercase** letters.
* **Ignores non-alphabetic** characters (numbers, symbols).
* **Wraps around** the alphabet (e.g., Z shifted by 1 becomes A).

### **Implementation Steps**

1. **Input Handling** - Takes a **plaintext message** (e.g., Hello!).
   * Takes an **integer key** (shift value).
2. **Encryption Process** - For each character:
   * If **uppercase (A-Z)**, shifts within A-Z.
   * If **lowercase (a-z)**, shifts within a-z.
   * Non-alphabetic characters remain unchanged.
3. **Output** - Prints the **ciphertext**.

## **Conclusion**

These three programs demonstrate essential C programming concepts:

* **String manipulation** (counting letters, replacing characters).
* **Mathematical computations** (Coleman-Liau index).
* **Conditional logic** (handling different cases in DNA and Caesar Cipher).