**––––**

**Cairo University**

**Faculty of Computers and Artificial Intelligence**

CS112: Structured Programming

Assignment 1

* **Submitted by:**

|  |  |
| --- | --- |
| ID | Members' names |
| 20231167  mostafaehabakl@gmail.com | Mostafa Ehab Mostafa Akl |
| 20231039  eng.gamalmegahed@gmail.com | Gamal Megahed Sayed Mohamed |
| 20231200  khattabadel112005@gmail.com | Walid Adel Mordy Rohyem |

* **Submitted to:**

Dr. Mohamed Elramly

**Cipher 1**

1. **Initialization**:
   * Define necessary variables including **SYSTEM\_COLOR** and **RESET\_COLOR**.
   * Define the uppercase alphabet string **upp\_alpha**.
2. **Welcome Message**:
   * Display a welcome message explaining the functionality of the program.
3. **Main Loop**:
   * Enter a continuous loop to allow multiple operations.
4. **Menu**:
   * Display a menu for the user to choose between encryption, decryption, or ending the program.
5. **Input Validation**:
   * Prompt the user for their choice and validate the input.
   * Repeat until a valid choice is made.
6. **Encryption**:
   * If the user chooses encryption:
     + Prompt the user to enter the keys **a** and **b**.
     + Validate the keys to ensure **a** is coprime with 26 and both keys are positive integers.
     + Prompt the user for the message to encrypt.
     + Convert the message to uppercase.
     + Encrypt each character of the message using the affine cipher formula **(ax + b) % 26**.
     + Output the encrypted message.
7. **Decryption**:
   * If the user chooses decryption:
     + Prompt the user to enter the keys **a**, **b**, and **c**.
     + Validate the keys to ensure **a** is coprime with 26, **c** is the multiplicative inverse of **a**, and all keys are positive integers.
     + Prompt the user for the message to decrypt.
     + Convert the message to uppercase.
     + Decrypt each character of the message using the affine cipher formula **c \* (y - b) % 26**.
     + Output the decrypted message.
8. **Ending the Program**:
   * If the user chooses to end the program, return 0 to terminate execution.
9. **Error Handling**:
   * Provide error messages for invalid input choices and guide the user to enter valid input.
10. **Loop Continuation**:

* After each operation, display a separator and return to the menu to allow further operations.

**Cipher 2**

1. **Initialization**:
   * Display a welcome message to the user introducing the Route Cipher.
2. **Main Loop**:
   * Enter a continuous loop to allow the user to encrypt, decrypt, or exit.
3. **Menu**:
   * Display options for the user to choose between encrypting, decrypting, or exiting the program.
4. **Input Validation**:
   * Prompt the user for their choice and validate the input.
   * Repeat until a valid choice is made.
5. **Encryption**:
   * If the user chooses encryption:
     + Prompt the user to enter the message to encrypt.
     + Convert the message to uppercase and remove non-alphabetic characters.
     + Prompt the user to enter the key (number of columns).
     + Validate the key to ensure it's a positive integer.
     + Determine the number of rows required based on the message length and key.
     + Create a matrix (board) to hold the message characters.
     + Fill the matrix by traversing in a spiral pattern and adding characters from the message.
     + Output the encrypted message.
6. **Decryption**:
   * If the user chooses decryption:
     + Prompt the user to enter the message to decrypt.
     + Convert the message to uppercase and remove non-alphabetic characters.
     + Prompt the user to enter the key (number of columns).
     + Validate the key to ensure it's a positive integer.
     + Determine the number of rows required based on the message length and key.
     + Create a matrix (board) to hold the message characters.
     + Fill the matrix in a spiral pattern, placing characters from the decrypted message.
     + Traverse the matrix row by row and extract characters to form the decrypted message.
     + Output the decrypted message.
7. **Ending the Program**:
   * If the user chooses to exit, break out of the loop and terminate the program.
8. **Error Handling**:
   * Provide error messages for invalid input choices and guide the user to enter valid input.
9. **Loop Continuation**:
   * After each operation, return to the menu to allow further operations.

**Cipher 3**

1. **Input**: User selects whether to encrypt or decrypt a message.
2. **If Encryption**: a. Initialize an empty string **encrypt\_result** to store the encrypted message. b. Define the uppercase alphabet string **upp\_alpha**. c. Prompt the user to enter the message. d. Convert the message to uppercase. e. For each character in the message: i. Find the character's index in **upp\_alpha**. ii. Replace the character with its corresponding character from **upp\_alpha** in reverse order (i.e., 25 - index). f. Output the encrypted message.
3. **If Decryption**: a. Decryption is essentially the same process as encryption since Atbash cipher is symmetric. b. Repeat steps 2b to 2f.
4. **Handling Errors**: a. Check if user input is valid (1 for encryption, 2 for decryption, 3 for exit). b. If invalid input is detected, prompt the user to input again until valid input is provided.
5. **Loop**: Repeat the process until the user chooses to exit.

**Cipher 4**

**Encryption Algorithm:**

Initialize Variables: Declare variables for the plaintext message, the encryption key, and the encrypted message.

Convert to Uppercase: Convert both the plaintext message and the key to uppercase for consistency.

Repeat Key: If the length of the key is shorter than the plaintext message, repeat the key to match the length of the plaintext message.

Encrypt Message:

Iterate through each character of the plaintext message.

If the character is alphabetic:

Calculate the shift value by subtracting 'A' from the sum of the corresponding plaintext character and key character (both converted to ASCII values).

Apply the modular operation (mod 26) to handle wraparound.

Convert the resulting value back to a character by adding 'A' to it.

Append the encrypted character to the encrypted message.

Return Encrypted Message.

Decryption Algorithm:

Initialize Variables: Declare variables for the encrypted message, the decryption key, and the decrypted message.

Convert to Uppercase: Convert both the encrypted message and the key to uppercase.

Repeat Key: If the length of the key is shorter than the encrypted message, repeat the key to match the length of the encrypted message.

**Decrypt Message:**

Iterate through each character of the encrypted message.

If the character is alphabetic:

Calculate the shift value by subtracting the key character (converted to ASCII) from the encrypted message character (also converted to ASCII).

Add 26 to handle negative values before taking the modulo 26.

Apply the modular operation (mod 26) to handle wraparound.

Convert the resulting value back to a character by adding 'A' to it.

Append the decrypted character to the decrypted message.

Return Decrypted Message.

**Cipher 5**

**Encryption (Baconian Cipher):**

Input: Message to be encrypted.

Convert Characters to Binary:

For each character in the message:

Subtract 1 from the character's ASCII value.

Convert the ASCII value to a 5-bit binary representation using bitset.

Append the binary representation to the ciphertext string.

Encode Binary Digits:

Replace each '0' with 'a' and each '1' with 'b'.

Output: Encrypted message.

**Decryption (Baconian Cipher):**

Input: Encrypted message.

Convert Characters to Binary:

For each character in the message:

If the character is 'a', replace it with '0'; if it's 'b', replace it with '1'.

Group Binary Digits:

Divide the binary string into groups of 5 bits.

Decode Binary Digits:

For each group of 5 bits:

Convert the binary representation to decimal.

Add 1 to the decimal value.

Convert the decimal value to ASCII character.

Append the decoded character to the plaintext string.

Output: Decrypted message.

**Cipher 6**

**Encryption (Simple Substitution Cipher):**

Input: Message to be encrypted, substitution key, and the alphabet.

Prepare Key: Create an array of size 26 to store the substituted alphabet.

Initialize it with the characters from the key.

Fill the remaining slots with the remaining alphabets in order.

Substitute Characters:

For each character in the message:

If the character is a space, leave it unchanged.

Otherwise, substitute it with the corresponding character from the key array.

Output: Encrypted message.

**Decryption (Simple Substitution Cipher):**

Input: Encrypted message, substitution key, and the alphabet.

Prepare Key: Create an array of size 26 to store the substituted alphabet.

Initialize it with the characters from the key.

Fill the remaining slots with the remaining alphabets in order.

Reverse Substitution:

For each character in the encrypted message:

If the character is a space, leave it unchanged.

Otherwise, find its index in the key array and substitute it with the corresponding alphabet.

Output: Decrypted message.

**Cipher 7**

**Cipher 8**

**Cipher 9**

**Cipher 10**

**Encryption:**

Remove Spaces: Remove any spaces from the message.

Divide the Message into Rails:

For each rail (row), iterate through the message characters, skipping characters based on a specific pattern determined by the key.

Append each character in the message to its corresponding rail.

Combine Rails: Concatenate the characters from each rail to form the encrypted message.

**Decryption:**

Initialize a Placeholder: Create an empty string with the same length as the encrypted message.

Divide the Encrypted Message into Rails:

For each rail (row), iterate through the encrypted message characters, again skipping characters based on the key pattern.

Place each character in the appropriate position in the placeholder string.

Read Off the Decrypted Message: Read the characters in the placeholder string from left to right to obtain the decrypted message.