**IMPLEMENTATION OF AND/XOR OPERATION ON EACH CHARACTER OF A STRING WITH 127**

**Objective**

The objective of this laboratory experiment is to implement bitwise **AND** and **XOR** operations on each character of a given string with the constant value **127**.

The experiment aims to demonstrate basic bitwise operations, their effects on ASCII characters, and their applications in encryption and data manipulation.

**THEORY**

**Bitwise Operations**

Bitwise operations operate at the binary level, manipulating individual bits of data. The primary operations used in this experiment are:

1. **Bitwise AND (&)**:
   * The AND operation compares each bit of two values and returns 1 if both bits are 1, otherwise it returns 0.
   * Example:

01001010 (ASCII of 'J' → 74)

01111111 (127 in binary)

------------

01001010 (Result → 74, no change)

* + Used in masking operations where certain bits are retained while others are cleared.

1. **Bitwise XOR (^)**:
   * The XOR operation compares each bit of two values and returns 1 if the bits are different, otherwise it returns 0.
   * Example:

01001010 (ASCII of 'J' → 74)

01111111 (127 in binary)

------------

00110101 (Result → 53)

* + Often used in encryption and obfuscation techniques, as XORing a value twice with the same key restores the original value.

**ASCII Representation**

Each character in a string is stored as an ASCII value (integer). When applying bitwise operations, we manipulate these numeric representations directly.

**ALGORITHM**

1. Take a string as input from the user.
2. Iterate through each character in the string.
3. Apply:
   * Bitwise AND (& 127) operation.
   * Bitwise XOR (^ 127) operation.
4. Store and display the results.

**INTERACTION WITH PROGRAM**

**Input:**

Enter a string: Hello123

**Output:**

Original String: Hello123

Bitwise AND with 127: Hello123

Bitwise XOR with 127: ~}|

**EXPLANATION OF OUTPUT**

1. **Bitwise AND (& 127)**
   * Since 127 is 01111111 in binary, ANDing it with any ASCII character does not change values within the standard ASCII range (0-127).
   * Thus, the output remains the same as the input.
2. **Bitwise XOR (^ 127)**
   * XOR flips bits where 127 has 1s.
   * This alters ASCII values, producing seemingly random characters, often used in encryption.

**APPLICATIONS**

* **Data Masking**: Bitwise operations help in hiding data in security applications.
* **Encryption**: XOR operation is used in simple encryption schemes (e.g., one-time pad).
* **Bit Manipulation**: Useful in low-level programming and performance optimization.

**CONCLUSION**

This experiment demonstrated how bitwise **AND** and **XOR** operations work on character strings. While AND retains the original values, XOR produces an encrypted-like output. Such operations are foundational in cryptography, security, and data processing.

**REFERENCES**

1. William Stallings, *Cryptography and Network Security*, Pearson
2. Nina Godbole, *Information Systems Security*, Wiley

**CODE**

#include <iostream>

#include <string>

using namespace std;

int main() {

string input;

// Taking user input

cout << "Enter a string: ";

getline(cin, input);

string andResult = "", xorResult = "";

// Applying bitwise AND and XOR with 127

for (char c : input) {

andResult += static\_cast<char>(c & 127);

xorResult += static\_cast<char>(c ^ 127);

}

// Displaying the results

cout << "\nOriginal String: " << input;

cout << "\nBitwise AND with 127: " << andResult;

cout << "\nBitwise XOR with 127: " << xorResult;

return 0;

}

**OUTPUT**

Enter a string: Hello123

Original String: Hello123

Bitwise AND with 127: Hello123

Bitwise XOR with 127: ~}|