Lab 5: Playfair Cipher

AI331: Information Security & Cryptography

Overview

In this lab you will implement the **Playfair cipher**—a classical digraph substitution cipher used historically for hand encryption. You will: (i) construct a 5×5 key square from a keyword, (ii) implement *encryption* and *decryption* with the three Playfair rules (row, column, rectangle), (iii) handle input normalization and padding, and (iv) test against provided vectors.

Learning objectives

- Understand the mechanics of a digraph substitution cipher.
- Practice secure handling of text normalization and edge cases.
- Gain confidence with unit testing via cryptographic test vectors.

Background

Alphabet: Use the English alphabet with **J merged into I** (i.e., the 25-letter alphabet A–Z without J).

Key square construction: Given a keyword (e.g., `MONARCHY`), remove non-letters, uppercase, merge J→ I, and remove duplicates preserving order. Fill the 5×5 square with the processed keyword, then the remaining letters A–Z (without J).

Preprocessing plaintext:

1. Keep only letters A–Z, uppercase, map J→ I.
2. Split into digraphs. If a pair would contain identical letters (e.g., `LL`), insert an `X` between them: `L X L`.
3. If the final block is a single letter, pad with `X`.

Encryption rules for a pair (a, b) with positions (r_a, c_a), (r_b, c_b) in the square:

1. Same row ($r_a = r_b$): replace each by the letter to its right (wrapping around).
2. Same column ($c_a = c_b$): replace each by the letter below it (wrapping around).
3. Rectangle: replace a with the letter in (r_a, c_b) and b with the letter in (r_b, c_a).

Decryption applies the inverse (left instead of right; up instead of down; and the same rectangle rule).

Tasks

Part A: Implement Playfair

Write a program in `C++` that:

1. Reads the *mode* (`enc` or `dec`), a *keyword*, and a *message* (plaintext or ciphertext). You may read from stdin or accept `--mode`, `--key`, `--text` flags.

2. Builds the 5×5 key square with J→ I merging.
3. Normalizes input as described above.
4. Encrypts or decrypts according to the Playfair rules.
5. Prints the resulting text as a single uppercase string (A–Z only).

Part B: Unit tests (required)

Verify your implementation using the following test vectors. Ensure your output *exactly* matches the ciphertext for encryption, and that decrypting that ciphertext returns the normalized plaintext (with J→ I and padding as per rules).

Keyword	Plaintext → Ciphertext
MONARCHY	`INSTRUMENTS` → `GATLMZCLRQXA`
PLAYFAIREXAMPLE	`HIDE THE GOLD IN THE TREE STUMP` → `BMODZBXDNABEKUDMUIXMMOUVIF`
SECURITY	`BALLOON` → `IBOQMPPO`
CRYPTOGRAPHY	`DEFEND THE EAST WALL OF THE CASTLE` → `EFIFLFPBIVFGZBYFQUUDKYGIYOZBMD`
KNOWLEDGE	`KICK` → `WCPE`

Note. After decryption, you may see inserted padding X's. Do not attempt to automatically strip padding; simply output the direct decryption result.

Part C: CLI & I/O format

Your program must support both modes below (choose at least one; supporting both is recommended):

- **Flags:** `./playfair --mode enc --key "MONARCHY" --text "INSTRUMENTS"`
- **Stdin:** read three lines: `mode`, `key`, `text`.

Edge cases & rules checklist

- Non-letters in the input are ignored (spaces, digits, punctuation).
- Map `J` to `I` everywhere.
- Insert `X` between duplicate letters in a pair (e.g., `BALLOON` → `BA LX LO ON` before encryption).
- If the final character is alone, pad with `X`.
- Wrap-around in rows/columns when shifting.

Starter code (C++)

Use this skeleton. Do not change function signatures; fill in the TODOs.

```
#include <bits/stdc++.h>
using namespace std;

/*
  Playfair Cipher - Starter Skeleton (NO SOLUTION CODE)
  -----------------------------------------------------
  Complete the TODOs below. Do NOT change function signatures.
  You must implement:
```

```
   - buildKeySquare
   - toDigraphs
   - encrypt
   - decrypt

  CLI usage (choose one style):
   1) Flags:
      ./playfair --mode enc --key "MONARCHY" --text "INSTRUMENTS"
   2) Stdin (three lines):
      enc
      MONARCHY
      INSTRUMENTS
*/

// --- Utilities you may use as-is ---
string normalize(const string &s) {
   string r;
   for (char ch : s) {
      if ('a' <= ch && ch <= 'z') r.push_back(ch - 'a' + 'A');
      else if ('A' <= ch && ch <= 'Z') r.push_back(ch);
   }
   for (char &ch : r) if (ch == 'J') ch = 'I'; // J -> I convention
   return r;
}

// --- TODO #1: Build the 5x5 key square and fill 'pos' map ---
// Requirements:
// - Use alphabet A..Z without 'J' (merge J->I already handled by normalize
   ).
// - Remove duplicate letters from keyword preserving order.
// - Fill remaining letters of alphabet (without J) afterwards.
// - Return a 5x5 array<char,5> grid, and fill 'pos' with positions for
   quick lookup.
array<array<char,5>,5> buildKeySquare(const string &keyword, map<char,pair<
   int,int>> &pos) {
   // TODO: construct grid and fill 'pos'
   throw runtime_error("TODO: implement buildKeySquare");
}

// --- TODO #2: Split plaintext into digraphs with Playfair rules ---
// Rules:
// - Work on normalized text (A-Z, J->I, no non-letters).
// - If a pair would be 'AA', insert 'X' between them: 'A X A' (then
   continue).
// - If final letter is alone, pad with 'X'.
vector<pair<char,char>> toDigraphs(const string &plaintext) {
   // TODO: create digraphs as per rules
   throw runtime_error("TODO: implement toDigraphs");
}

// --- TODO #3: Encrypt using Playfair rules ---
// For each pair (a,b) at positions (ra,ca), (rb,cb):
// - Same row: replace with letters to the RIGHT (wrap around).
// - Same column: replace with letters BELOW (wrap around).
// - Rectangle: replace a with (ra,cb) and b with (rb,ca).
string encrypt(const string &plaintext, const array<array<char,5>,5> &grid,
    map<char,pair<int,int>> &pos) {
   // TODO: implement encryption using toDigraphs(...) and 'grid'/'pos'
```

```
    throw runtime_error("TODO: implement encrypt");
}

// --- TODO #4: Decrypt using inverse rules ---
// Same row -> shift LEFT; same column -> shift UP; rectangle rule is
    symmetric.
string decrypt(const string &ciphertext, const array<array<char,5>,5> &grid
    , map<char,pair<int,int>> &pos) {
    // TODO: implement decryption
    throw runtime_error("TODO: implement decrypt");
}

int main(int argc, char** argv) {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    string mode, key, text;

    // --- Simple CLI parsing ---
    if (argc > 1) {
        for (int i = 1; i < argc; ++i) {
            string a = argv[i];
            if (a == "--mode" && i+1 < argc) mode = argv[++i];
            else if (a == "--key" && i+1 < argc) key = argv[++i];
            else if (a == "--text" && i+1 < argc) text = argv[++i];
        }
    }
    if (mode.empty()) {
        // Fallback: read three lines from stdin
        if (!getline(cin, mode)) return 0;
        getline(cin, key);
        getline(cin, text);
    }

    try {
        map<char,pair<int,int>> pos;
        auto grid = buildKeySquare(key, pos);

        if (mode == "enc") {
            cout << encrypt(text, grid, pos) << "\n";
        } else if (mode == "dec") {
            cout << decrypt(text, grid, pos) << "\n";
        } else {
            cerr << "Invalid mode. Use enc|dec.\n";
            return 1;
        }
    } catch (const exception &e) {
        cerr << "[ERROR] " << e.what() << "\n";
        cerr << "Hint: Complete the TODOs in buildKeySquare / toDigraphs /
            encrypt / decrypt.\n";
        return 2;
    }
    return 0;
}
```

What to submit

- Source code files (e.g., `playfair.cpp`) and a short `README.md` explaining build & run instructions.
- A text file `results.txt` with your outputs for all test vectors in Part B (both encryption and decryption).