CSCI 113 Project – Rubik’s Cube Solver

# Code Analysis and Implementation

## Core Data Structures

* Used a 3D char array cube[6][3][3] to represent the Rubik's cube
* Each face represented by a 2D array with constant dimensions N\_ROWS = 3, N\_COLS = 3
* Face indices: TOP = 0, LEFT = 1, FRONT = 2, RIGHT = 3, BACK = 4, BOTTOM = 5

## Function Overview

### Core Movement Functions

* moveU(), moveD(), moveF(), moveB(), moveR(), moveL(): Implement basic cube rotations
* rotateFaceClockwise(), rotateFaceCounterClockwise(): Handle face rotation mechanics
* Each movement function maintains cube state consistency by tracking edge pieces

### Solving Algorithm Functions

* solveF2l(): Implements First Two Layers method
* solveOll(): Handles Orientation of Last Layer
* solvePll(): Manages Permutation of Last Layer
* solveWhiteCross(): Creates initial white cross pattern

### Helper Functions

* isValidMove(): Validates input moves
* applyMove(): Executes validated moves
* displayCube(): Renders current cube state
* isSolved(): Verifies solution completion

## Algorithm Implementation Details

### F2L Implementation

* Uses pair-wise corner-edge matching
* Implements different algorithms based on piece positions

### Movement Validation

* Ensures move notation follows standard conventions
* Handles prime (') and double (2) moves

# Results and Testing

## Test Cases and Screenshots

### Initial State

W W W

W W W

W W W

O O O G G G R R R B B B

O O O G G G R R R B B B

O O O G G G R R R B B B

Y Y Y

Y Y Y

Y Y Y

The first face is top, then from left to right: left, front, right, back, then the last face is the bottom face.

**Main Menu**

Rubik’s Cube Solver is a console based c++ program used to solve a Cube when user inputs a scramble the main options in the menu are:

1.Scrambling the cube:

Lets user input a scramble and program simulates scrambling process.

2. Solve up to F2L

* Solve White Cross: Aligns white edge pieces to form a cross on the white face.
* Solve White Corners: Places white corner pieces correctly.
* Solve Middle Layer: Positions and orients edge pieces in the middle layer

**3. Solve Last Layers**

Orients and positions pieces on the last layer to complete the cube.

**4. Display Cube**

Prints the current state of cube in 3d folded out form.

**5. Check if the Cube is Solved**

Validates whether all sides of the cube are uniform, indicating a solved cube.

**6. Reset the Cube**

Resets the cube to its original solved state.

**7. Exit**

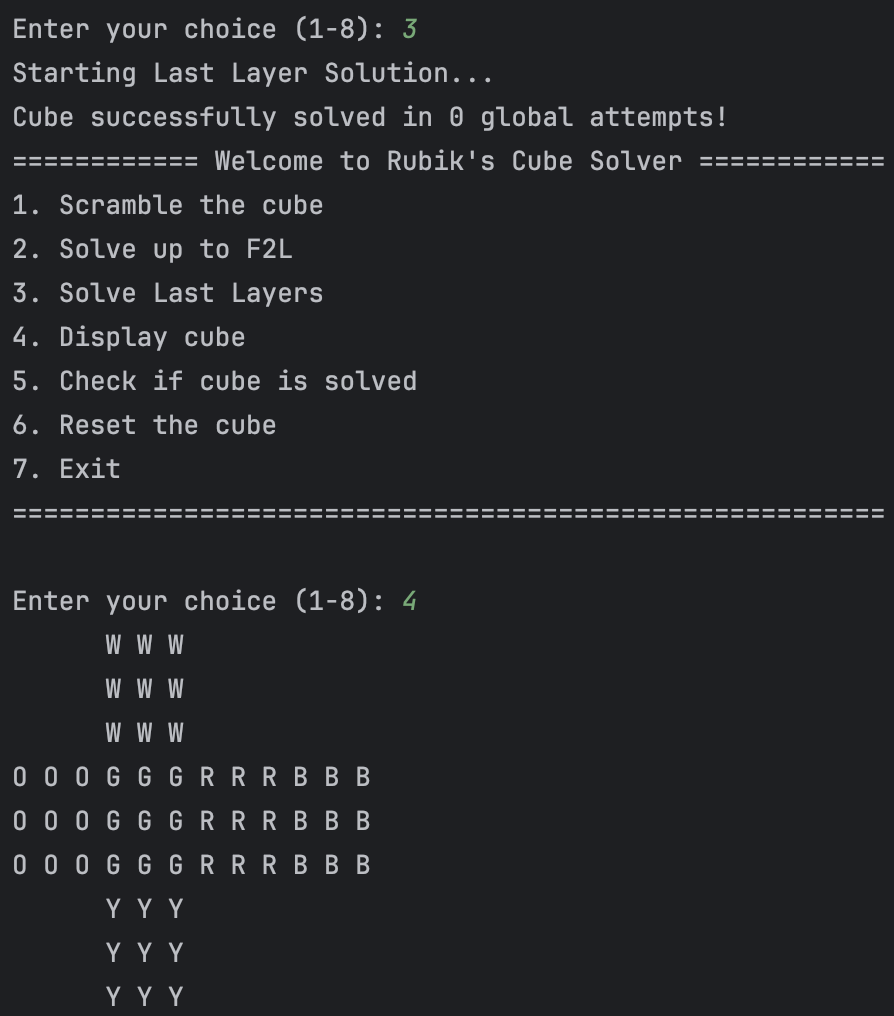
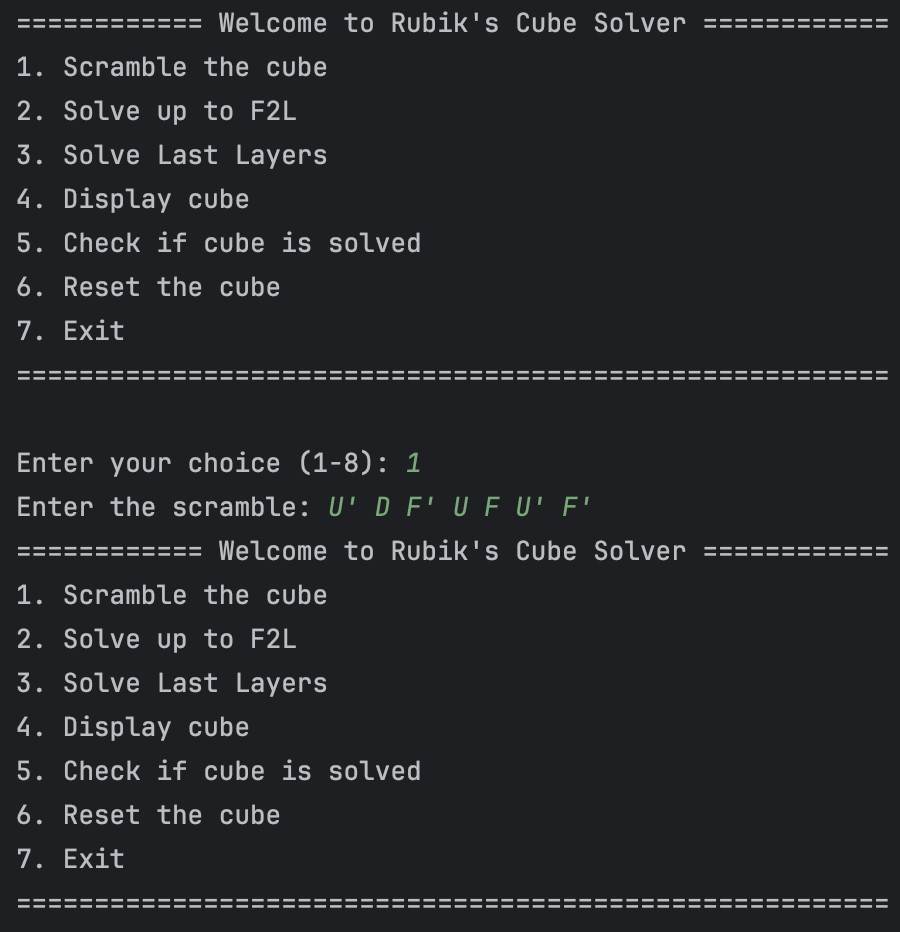
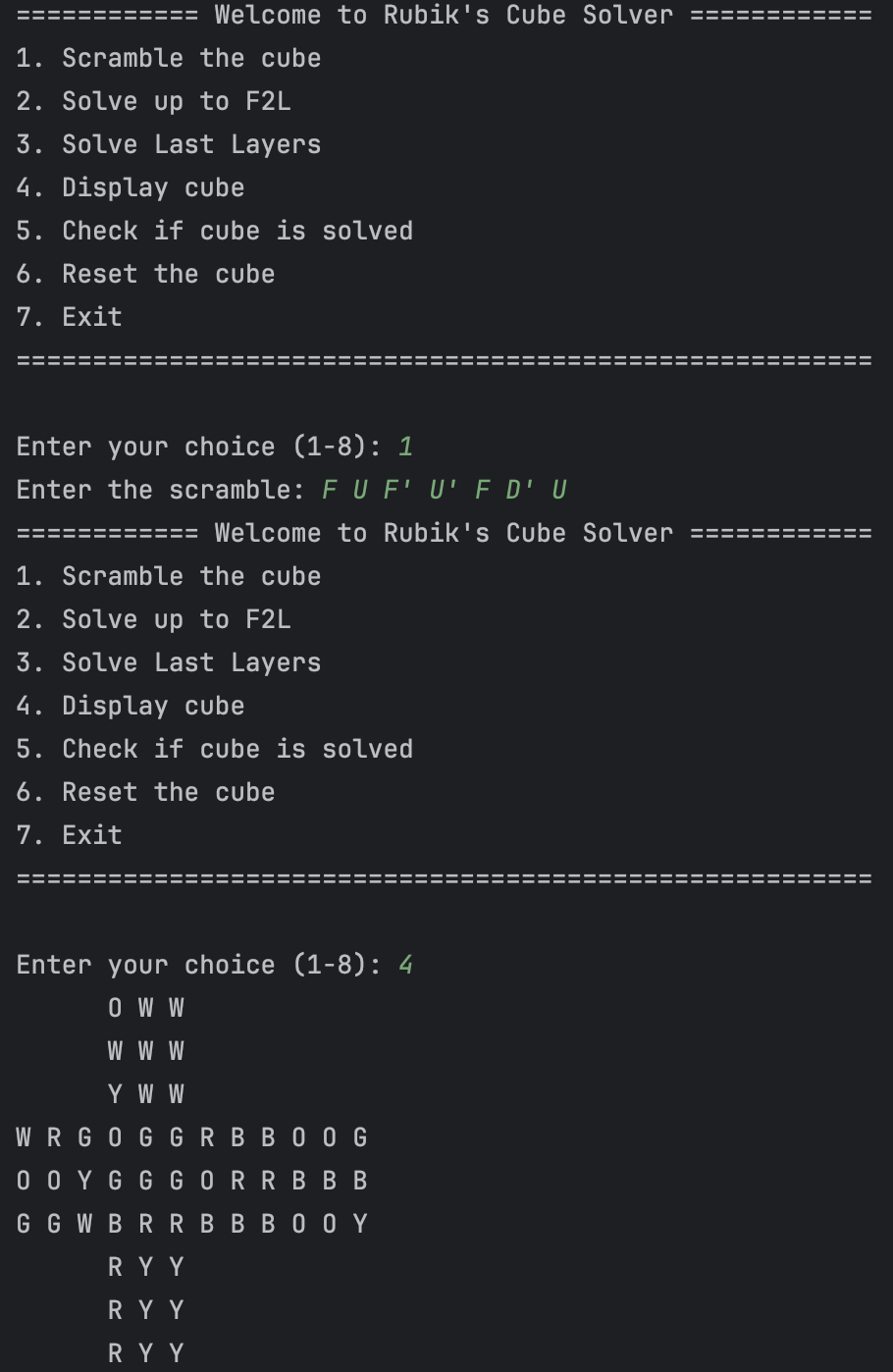
Ends the program with a Goodbye message.

**User Input Handling**

Input Validation: Ensures users provide valid numeric input between 1 and 7. Invalid inputs prints out error message, asking user to try again.

Error Recovery: Uses input stream error handling (cin.clear() and cin.ignore()) to manage invalid inputs without crashing.

### Solution Progress



## Problem Resolution

### Face Rotation Challenge

* Issue: Initial face rotation algorithm caused edge piece misalignment
* Solution: Implemented temporary array storage for edge preservation

void rotateFaceClockwise(char face[N\_ROWS][N\_COLS]) {

char temp[N\_ROWS][N\_COLS];

for (int i = 0; i < N\_ROWS; i++) {

for (int j = 0; j < N\_COLS; j++) {

temp[i][j] = face[i][j];

}

}

...

}

### F2L Pair Insertion

* Issue: Corner-edge pairs sometimes became misaligned during insertion
* Solution: Added validation checks and corrective algorithms

# Task Distribution and Coordination

## Task Assignment Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Assignee** | **Status** | **Notes** |
| Core Data Structure Design | Hassan | Complete | Implemented cube representation |
| Basic Move Functions | Hasnain | Complete | U, D, F, B, L, R moves |
| F2L Implementation | Hasnain | Complete | Corner-edge pairing |
| OLL Implementation | Hassan | In progress | Basic patterns complete |
| PLL Implementation | Hasnain | Complete | Working on algorithms |

## Coordination Method

* Used Git for version control and code sharing
* Maintained detailed comments for code handoffs

## Areas for Improvement

### Algorithm Optimization

* Current F2L implementation could be optimized for fewer moves
* PLL algorithm set could be expanded

### Error Handling

* Add more robust input validation
* Implement recovery mechanisms for invalid states

### User Interface

* Consider adding visual cube representation
* Implement move suggestion system

# Function Documentation

## Essential Functions

### void solveF2l()

#### Purpose

Implements the First Two Layers solving method

#### Input

None (operates on global cube state)

#### Output

None (modifies cube state)

#### Process

Iteratively solves corner-edge pairs

### void moveU(), moveD(), moveF(), moveB(), moveR(), moveL()

#### Purpose

Execute basic cube rotations

#### Input

None

#### Output

None (modifies cube state)

#### Process

Rotates face and adjusts adjacent edges

### bool isValidMove(const string& MOVE)

#### Purpose

Validates move notation

#### Input

Move string

#### Output

Boolean indicating validity

#### Process

Checks against valid move patterns

### void displayCube()

#### Purpose

Renders current cube state

#### Input

None

#### Output

Console visualization of cube

#### Process

Prints face arrays in 2D layout