Name: Nitin Choudhary
Roll no: 50
Class: TY-IT A B3

8-Puzzle using Simple Hill Climbing:

```cpp
#include <iostream>
#include <vector>

using namespace std;

// Function to calculate the number of misplaced tiles
int misplaced_tiles(vector<vector<int>> &state, vector<vector<int>>
&goal_state)
{
    int misplaced_count = 0;
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (state[i][j] != goal_state[i][j])
            {
                misplaced_count++;
            }
        }
    }
    return misplaced_count;
}

// Function to get next possible states
void get_next_states(vector<vector<int>> &state, vector<vector<vector<int>>>
&next_states)
{
    int moves[4][2] = {{0, 1}, {0, -1}, {1, 0}, {-1, 0}};
    int zero_row, zero_col;

    // Find the position of the empty (0) tile
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (state[i][j] == 0)
            {
                zero_row = i;
                zero_col = j;
            }
        }
    }

    // Generate next states by swapping the empty tile with its neighbors
    for (int i = 0; i < 4; i++)
    {
        int new_row = zero_row + moves[i][0];
        int new_col = zero_col + moves[i][1];

        if (new_row >= 0 && new_row < 3 && new_col >= 0 && new_col < 3)
        {
            // Create a copy of the current state
            vector<vector<int>> new_state = state;

            // Swap the empty tile with the neighbor
            swap(new_state[zero_row][zero_col],
new_state[new_row][new_col]);

            next_states.push_back(new_state);
        }
    }
}

int calculate_f(vector<vector<int>> &state, vector<vector<int>> &goal_state)
{
    return misplaced_tiles(state, goal_state);
}

int main()
{
    vector<vector<int>> initial_state = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 0}};

    vector<vector<int>> goal_state = {
        {1, 2, 3},
        {4, 8, 5},
        {0, 7, 6}};

    cout << "Initial State:" << endl;
    for (int i = 0; i < 3; i++)
    {
```

```cpp
        for (int j = 0; j < 3; j++)
        {
            cout << initial_state[i][j] << " ";
        }
        cout << endl;
    }

    int initial_f = calculate_f(initial_state, goal_state);
    cout << "f(x) = " << initial_f << endl;

    vector<vector<vector<int>>> next_states;
    get_next_states(initial_state, next_states);

    int best_f = initial_f;
    int best_index = -1;

    cout << "\nBest Next State:" << endl;
    for (int i = 0; i < next_states.size(); i++)
    {
        int f = calculate_f(next_states[i], goal_state);
        if (f < best_f)
        {
            best_f = f;
            best_index = i;
        }
    }

    if (best_index != -1)
    {
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                cout << next_states[best_index][i][j] << " ";
            }
            cout << endl;
        }
        cout << "f(x) of Best Next State: " << best_f << endl;
    }
    else
    {
        cout << "No better state found." << endl;
    }

    return 0;
```

```
}
```

Output:

```
Initial State:
1 2 3
4 5 6
7 8 0
f(x) = 5

Best Next State:
1 2 3
4 5 0
7 8 6
f(x) of Best Next State: 4
```