# CSCI 135 Software Design and Analysis, C++
# Homework 8
# Solution

Saad Mneimneh
Computer Science
Hunter College of CUNY

**Problem 1: Two-dimensional arrays and the 15 puzzle**
The 15 puzzle consist of 15 pieces numbered 1 to 15, arranged on a 4x4 board with one free spot. The free spot allows to move the pieces to the left, right, up, and down. The goal of the puzzle is to put it in the "solved" state:

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | |

Consider the following class in a file called puzzle.h to represent the 15 puzzle:

```
#ifndef _15PUZZLE
#define _15PUZZLE

#include <iostream>
#include <string>
using std::cout;
using std::string;

class Puzzle {
 private:
  int a[4][4];
  int i, j;    //row and column of the empty spot
  int correct; //number of pieces that have the correct position

 public:
  Puzzle();
  Puzzle(const string& s); //passed by reference to avoid copy
  void scramble(int n);
  bool move(char c);
  bool move(int n);
  bool solved() {return correct=15;}  //obvious!
  void display();
};

#endif
```

**PART I**
For this part, you will implement the Puzzle class in a file called puzzle.c (or puzzle.cpp).

(a) Implement the default constructor to put the puzzle in its solved state. This also means that $i$ and $j$ are both set to 3, and correct is set to 15.

(b) Implement the second constructor in the following way: First the puzzle is initialized in the same way as in the default constructor. Second, the string $s$ is interpreted as a sequence of characters, obtained as usual as $s[0]$, $s[1]$, $s[2]$, etc... The length of the string can be determined using the member function length, e.g. s.length(). The constructor then calls the move function on each of the characters of $s$.

(c) Implement the scramble function to repeatedly generate a random character in the set $\{'l','r','u','d'\}$ (these stand for left, right, up, and down), and call the move function on that character. There should be $n$ successful moves in total (the move function returns true upon success and false otherwise).
(d) The move (with char parameter) is the heart of this game. It should be implemented as follows.

- The character is interpreted as described above. If the character is not in the set $\{'l','r','u','d'\}$, then the move is not performed and the function returns false.
- Since the position of the empty spot is known at all times, it is easy to determine if the given move is feasible. For example, 'l' means move something to the left. Therefore, there must be a piece to the right of the empty spot.
- If the move is feasible, perform it by updating the array, the position of the empty spot, and the number of correct positions.
- The function returns true if the move is performed and false otherwise.

(d) The move (with int parameter) determines if $n$ is to the left, right, above, or below the empty spot. If not, the move is not performed and the function returns false. Otherwise, the function returns the result of the corresponding char move.

(e) Implement the display function to output the current state of the 15 puzzle in a nice way.

**PART II**
For this part, you will implement the 15 puzzle game in a file called main.c (or main.cpp).

(a) Initialize a puzzle using the default constructor. Then ask the user to input a number $n$, and scramble the puzzle with $n$ as the parameter to the function. Finally display the state of the puzzle.

(b) Following part (a), repeatedly ask the user to input a string. If the first character of the string is not a digit, then call the char move function on that character. If the first character of the string is a digit, then consider at most two

characters to determine the number, and call the int move function on that number. After every successful move, display the state of the puzzle. The program should stop when the puzzle is solved (or when the user terminates it of course).

**Solution**:

```
#include <iostream>
#include <string>
#include <cstdlib>

using std::cout;
using std::cin;
using std::string;

class Puzzle {
  int a[4][4];
  int i;
  int j;
  int correct;

  void init();
  bool spotCorrect(int i, int j) {return (a[i][j]==4*i+j+1);}

 public:
  Puzzle();
  Puzzle(const string& s);
  bool move(char c);
  bool move(int n);
  void scramble(int n);
  bool solved() {return (correct==15);}
  void display();
};

void Puzzle::init() {
  for (int row=0; row<4; row++)
    for (int col=0; col<4; col++)
      a[row][col]=row*4+col+1;  //should be self explanatory
  i=j=3;
  correct=15;
}

Puzzle::Puzzle() {
  init();
}

Puzzle::Puzzle(const string& s) {
  init();
  for (int i=0; i<s.length(); i++)
    move(s[i]);
}
```

```
void Puzzle::scramble(int n) {
  char b[4]={'u', 'd', 'l', 'r'};
  int i=0;
  while (i<n)
    if (move(b[rand()%4])) //if move succeeds, increment i
      i++;
}

bool Puzzle::move(char c) {
  if (c=='u' && i<3) {  //empty spot is not in last row
    if (spotCorrect(i+1,j)) //spot was correct
      correct--;
    a[i][j]=a[i+1][j];
    if (spotCorrect(i,j)) //spot is now correct
      correct++;
    a[i+1][j]=16;
    i++;
    return true;
  }

  if (c=='d' && i>0) {  //empty spot is not in first row
    if (spotCorrect(i-1,j))
      correct--;
    a[i][j]=a[i-1][j];
    if (spotCorrect(i,j))
      correct++;
    a[i-1][j]=16;
    i--;
    return true;
  }

  if (c=='l' && j<3) {  //empty spot is not in last column
    if (spotCorrect(i,j+1))
      correct--;
    a[i][j]=a[i][j+1];
    if (spotCorrect(i,j))
      correct++;
    a[i][j+1]=16;
    j++;
    return true;
  }

  if (c=='r' && j>0) {  //empty spot is not in first column
    if (spotCorrect(i,j-1))
      correct--;
    a[i][j]=a[i][j-1];
    if (spotCorrect(i,j))
      correct++;
    a[i][j-1]=16;
    j--;
    return true;
  }
```

```
  return false;
}

bool Puzzle::move(int n) {
  if (i<3 && a[i+1][j]==n)
    return move('u');
  if (i>0 && a[i-1][j]==n)
    return move('d');
  if (j<3 && a[i][j+1]==n)
    return move('l');
  if (j>0 && a[i][j-1]==n)
    return move('r');
  return false;
}

void Puzzle::display() {
  for (int row=0; row<4; row++) {
    for (int col=0; col<4; col++) {
      if (a[row][col]<10)
        cout<<' ';
      if (a[row][col]==16)
        cout<<"   ";
      else
        cout<<a[row][col]<<' ';
    }
    cout<<'\n';
  }
  cout<<'\n';
}

int main() {       //this is a simplified main
  srand(time(0));
  int i;
  Puzzle p;
  p.scramble(10);
  p.display();
  while (!p.solved()) {
    cin>>i;
    p.move(i);
    p.display();
  }
}
```