PF-LAB PROJECT

SEMESTER 1

**Void Initial:**

* *Randomly initializing the initial board to an array using srand().*
* *The initial state doesn’t repeat any value.*

**CODE:**

void initial\_board()

{

int temp;

int ar[25], ar1[5][5];

bool flag;

for (int i = 0; i < 25; i++)

{

do

{

temp = rand() % 25;

flag = true;

for (int j = 0; j < i; j++)

{

if (ar[j] == temp)

{

flag = false;

}

}

} while (!flag);

ar[i] = temp;

}

int p = 0;

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

ar1[i][j] = ar[p];

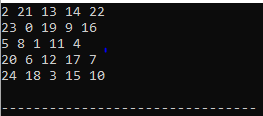
p++;

}

}

}

**OUTPUT:**



**Void Final :**

* *Same as above this function initializes a random goal state.*
* *The goal state is also without repetition and it stays static throughout the game.*

**Code:**

void final\_board()

{

int temp;

int ar[25];

bool flag;

for (int i = 0; i < 25; i++)

{

do

{

temp = rand() % 25;

flag = true;

for (int j = 0; j < i; j++)

{

if (ar[j] == temp)

{

flag = false;

}

}

} while (!flag);

ar[i] = temp;

}

int p = 0;

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

ar2[i][j] = ar[p];

p++;

}

}

}

**Bool Is\_solvable:**

* *Takes initial and final as arguments and tell if final board can be reached or not.*
* *Here I calculated the inversions of initial and final board and compared then to reach the solution.*

**Code:**

//solvable

bool is\_solvable(int ar1[5][5], int ar2[5][5])

{

int temp\_1[25], temp\_2[25];

int k = 0, p = 0, count1 = 0, count2 = 0, sum1 = 0, sum2 = 0;

bool res = false;

//first bogus array

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

temp\_1[k] = ar1[i][j];

k++;

}

}

// second bogus array

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

temp\_2[p] = ar2[i][j];

p++;

}

}

//inversions of first bogus array

for (int i = 0; i < 25; i++)

{

count1 = 0;

for (int j = 0; j < 25; j++)

{

if (temp\_1[i] > temp\_1[j])

{

count1++;

}

}

sum1 += count1;

}

//inversions of second bogus array

for (int i = 0; i < 25; i++)

{

count2 = 0;

for (int j = 0; j < 25; j++)

{

if (temp\_2[i] > temp\_2[j])

{

count2++;

}

}

sum2 += count2;

}

// solvable or not

if ((sum1 % 2 == 0 && sum2 % 2 == 0) || (sum1 % 2 != 0 && sum2 % 2 != 0))

{

res = true;

return res;

}

else

return res;

}

**Void Legal Moves:**

* *This function takes the initial array as an argument and tells the possible legal moves on the board at that state.*
* *It also restricts the user to make any illegal move on the board.*

**Code:**

void legal\_moves(int array[5][5])

{

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

if (array[i][j] == 0)

{

if (i == 0 && j == 0)

{

cout << "Legal Moves at this position are:\n" << "LEFT\n" << "UP\n";

}

else if ((i == 0 && j == 1) || (i == 0 && j == 2) || (i == 0 && j == 3))

cout << "Legal Moves at this position are:\n" << "RIGHT\n" << "LEFT\n" << "UP\n";

else if ((i == 0 && j == 4))

cout << "Legal Moves at this position are:\n" << "RIGHT\n" << "UP\n";

else if ((i == 1 && j == 0) || (i == 2 && j == 0) || (i == 3 && j == 0))

cout << "Legal Moves at this position are:\n" << "RIGHT\n" << "UP\n" << "DOWN\n";

else if ((i == 1 && j == 4) || (i == 2 && j == 4) || (i == 3 && j == 4))

cout << "Legal Moves at this position are:\n" << "LEFT\n" << "UP\n" << "DOWN\n";

else if ((i == 4 && j == 0))

cout << "Legal Moves at this position are:\n" << "LEFT\n" << "DOWN\n";

else if ((i == 4 && j == 1) || (i == 4 && j == 2) || (i == 4 && j == 3))

cout << "Legal Moves at this position are:\n" << "RIGHT\n" << "LEFT\n" << "DOWN\n";

else if (i == 4 && j == 4)

cout << "Legal Moves at this position are:\n" << "RIGHT\n" << "DOWN\n";

else

cout << "Legal Moves at this position are:\n" << "RIGHT\n" << "LEFT\n" << "UP\n" << "DOWN\n";

}

}

}

}

Void \_Save:

* *This function allows user to save the game and continue again if he wants to.*
* *Here I made a handler and a text file to store the array values to a file.*
* *The game stops here and asks user if he wants to continue.*

**Code:**

void save(int array1[5][5],int array2[5][5])

{

//initial array saver

ofstream saver1 ;

saver1.open("saveini.txt",ios\_base::out) ;

if(!saver1)

{

cout << "Error" ;

}

else

{

for(int i=0;i<5;i++)

{

for(int j=0;j<5;j++)

{

saver1<< permanent1[i][j] <<" ";

}

}

saver1.close() ;

}

//final array saver

ofstream saver2 ;

saver2.open("savefinal.txt",ios\_base::out) ;

if(!saver2)

{

cout << "Error" ;

}

else

{

for(int i=0;i<5;i++)

{

for(int j=0;j<5;j++)

{

saver2<< permanent2[i][j] <<" ";

}

}

saver2.close() ;

}

}

Void\_Loader:

* Loads the value of initial and final board to the arrays again to continue the game.
* The atoi functions has been used to convert string type data that’s being read from file to integer to store it in the array.

**Code:**

void load(int array1[5][5],int array2[5][5])

{

//initial array loader

string \_char1 ;

ifstream loader1 ;

loader1.open("saveini.txt",ios::in) ;

if(!loader1)

{

cout << "Error" ;

}

else

{

for(int i=0;i<5;i++)

{

for(int j=0;j<5;j++)

{

loader1 >> \_char1 ;

if(\_char1 !=" ")

{

permanent1[i][j] = atoi(\_char1.c\_str ( ) );

}

}

}

}

//final array loader

string \_char2 ;

ifstream loader2 ;

loader2.open("savefinal.txt",ios::in) ;

if(!loader2)

{

cout << "Error" ;

}

else

{

for(int i=0;i<5;i++)

{

for(int j=0;j<5;j++)

{

loader2 >> \_char2 ;

if(\_char2 !=" ")

{

permanent2[i][j] = atoi(\_char2.c\_str ( ) );

}

}

}

}

}

USMAN AWAN 22F-3378

# Void is\_goal()

* This function will compare each index of the initial board with each index of the final board
* After comparing it will return true or false based on the condition that if each index of the initial board is equal to each index of the final board it will return true to the main function otherwise it will return false.

bool is\_goal()

{

bool is\_goal\_res = true;

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

if (permanent1[i][j] == permanent2[i][j])

{

is\_goal\_res = true;

}

else

is\_goal\_res = false;

return false;

}

}

return true;

}

# Void make\_move()

* For this function, I declared two variables named r for row and c for column.
* Then I checked each index of the initial board by loop, to check where there is 0 present in the board and put the row index I of that index in r and column index j in c variable.
* Prupose of doing this is to identify where we will start making moves as we know that we start by the empty space.
* Then we will use that r and c index to swap the values of the index.
* For the process of swapping I used the arrow keys and to use arrow keys I declared these arrow keys as constant values by specifying them an ascii before main().
* When ever I will press arrow key using the \_getch(). Firstly it will determine that wheather the pressed key is in the arrow key asci that I declared globally, secondly if it is a arrow key it will then check that which arrow key was pressed among the 4 contant keys. Then it will swap accordingly.
* I also used the save file function that if the user presses escape key then it will save the program and will ask the user to select y to load the program or n to continue.
* I also used this function to store the value of pressed keys and store them in a string.

void make\_move()

{

int r = 0, c = 0;

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

if (permanent1[i][j] == 0)

{

r = i;

c = j;

}

}

}

char decision;

unsigned char char\_read = \_getch();

if (char\_read == ARROW\_KEY\_CHARACTER)

{

unsigned char arrow\_read = \_getch();

switch (arrow\_read)

{

case ARROW\_KEY\_UP:

if (r == 4 && (c == 0 || c == 1 || c == 2 || c == 3 || c == 4))

{

cout << "Error Illegal move\n";

}

else

{

swap(permanent1[r][c], permanent1[r + 1][c]);

path += "U, ";

}

break;

case ARROW\_KEY\_DOWN:

if (r == 0 && (c == 0 || c == 1 || c == 2 || c == 3 || c == 4))

{

cout << "Error Illegal move\n";

}

else

{

swap(permanent1[r][c], permanent1[r - 1][c]);

path += "D, ";

}

break;

case ARROW\_KEY\_LEFT:

if ((r == 0 || r == 1 || r == 2 || r == 3 || r == 4) && c == 4)

{

cout << "Error Illegal move\n";

}

else

{

swap(permanent1[r][c], permanent1[r][c + 1]);

path += "L, ";

}

break;

case ARROW\_KEY\_RIGHT:

if ((r == 0 || r == 1 || r == 2 || r == 3 || r == 4) && c == 0)

{

cout << "Error Illegal move\n";

}

else

{

swap(permanent1[r][c], permanent1[r][c - 1]);

path += "R, ";

}

break;

}

}

else

{

if (char\_read == escape)

{

save(permanent1, permanent2);

cout << endl;

cout << "Your game has been saved\n";

cout << "Do you want to continue? Y or N?\n";

decision = \_getch();

if (decision == 'Y')

{

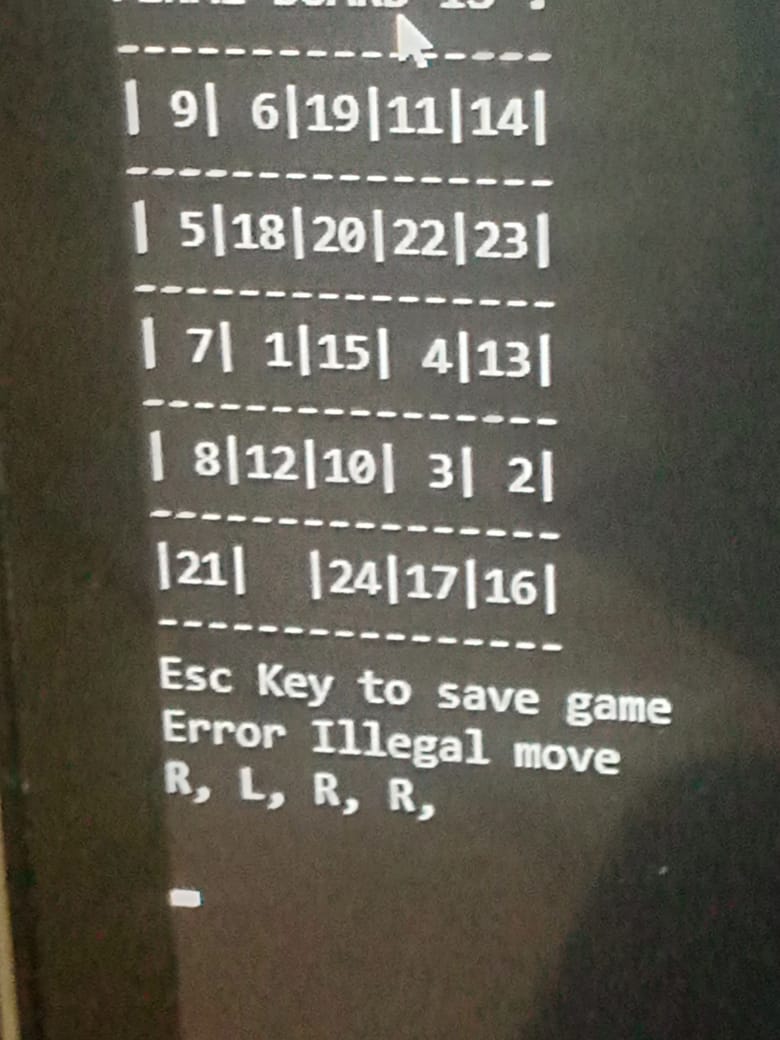
load(permanent1, permanent2);

}

}

}

}



# void print\_path()

* This function is associated with the make move function
* Purpose of this function is to print the path of the moves that the user performed to reach the goal board.
* In function make\_move, each move is being stored in the variable path
* And when print\_path function is called it will return al the moves that the user performed after the goal state has been reached.

void print\_path()

{

for (int i = 0; path[i] != '\0'; i++)

cout << path[i];

}

# 

# Main()

* In main(), firstly the initial\_board() is called and then the final\_board and stores the values in both arrays named ar1 and ar2.
* Now two arrays permanent1 and permanent2 are used to store the values of the ar1 and ar2 as the main arrays for the whole program.
* Now a loop is used to check wheather the initial\_board is reachable to final\_board.If it is reachable to goal\_board then it will print goal is reachable and will move towards the next function which is init().
* Now the init and finale board is called and whole board is printed on the console.
* In while(1) loop, the program keeps checking what are the legal moves that a user can perform.
* And also perform moves .
* Now while(1) loop is used to keep taking values from the user until there is a statement inside the loop body which is that when the is\_goal function returns true stops the program and prints the moves thaht user took to reach goal state.

int main()

{

srand(time(0));

initial\_board();

cout << endl << endl;

final\_board();

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

permanent1[i][j] = ar1[i][j];

}

}

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

{

permanent2[i][j] = ar2[i][j];

}

}

while (!is\_solvable(permanent1, permanent2))

{

if (is\_solvable(permanent1, permanent2))

cout << "THE GOAL IS REACHABLE\n";

}

init();

cout << endl << endl;

finale();

legal\_moves(permanent1);

while (1)

{

cout << "MAKE YOUR MOVE\n";

init();

legal\_moves(permanent1);

finale();

cout << "Esc Key to save game\n";

make\_move();

if (is\_goal())

{

cout << "\tYou won the game congrats\n";

cout << "\t\tYou did the following moves\n";

print\_path();

break;

}

Sleep(2000);

cout << endl << endl;

system("cls");

}

}

