PROGRAMMING FUNDAMENTALS - (CT-175).

PROJECT REPORT

TITLE: "NUMBER SHIFTING GAME".

GROUP NAME: BYTE BLASTERS

GROUP MEMBERS:

- M RAYYAN LODHI (CTCY-010).
- SYED M FAIZAN HYDER (CTCY-026).
- SYED M SAQIB FAHEEM (CTCY-011).

TEACHER: MISS SAMIA MASOOD AWAN.

HANDED IN: 15- JAN -2024.

1.)PROBLEM STATEMENT:

We have made a C program that allows the user to shift the numbers up ,down , left and right using arrow keys. The program uses a function to moves numbers in up and down , left and right to arrange numbers in sequencial order.

2.)PROBLEM DESCRIPTION:

The program we have made is a simple game that presents the C code implements a console-based sliding puzzle game, creating a 4x4 matrix with randomly arranged numbers from 1 to 15 and an empty cell. Players can move numbers within the matrix using arrow keys, attempting to achieve a winning state where the numbers are in order. The program displays game rules and allows for multiple gaming sessions, notifying players of their outcome – either winning or losing – based on the number of moves taken. Players can exit the game at any time by pressing 'E' or 'e'.

Pseudo Code Of the Program

1. **Initialization:**

    - Initialize an empty 4x4 matrix and auxiliary arrays (ar and arr) to track numbers.

2. **Random Number Placement:**

- Randomly arrange numbers from 1 to 15 in the matrix, using (ar) to ensure unique placements.
- Leave one cell in the matrix as empty (represented by 0).

3. **Matrix Display:**

- Create a graphical representation to display the current state of the matrix.

4. **Winning Check:**

- Check if the player has won by verifying if the matrix is in the correct order (numbers 1 to 15).

5. **Element Swapping:**

- Implement swapping of elements in the matrix based on the direction chosen by the player (up, down, left, right).

6. **Game Rules Display:**

- Display game rules, including instructions on how to move, win, and exit the game.

7. **Main Loop:**

- Enter a main loop for multiple game sessions.

8. **Matrix Regeneration:**

- Generate a new matrix for each game session.

9. **Rule Display:**

- Display game rules for the current session.

10. **Gameplay Loop:**

- Enter a loop where the player attempts to solve the puzzle within a limited number of moves.

11. **Outcome Check:**

- Check if the player has won or lost based on the number of moves taken.

12. **Outcome Display:**

- Inform the player of the outcome (winning or losing).

13. **Exit Option**:

- Allow the player to exit the game at any time.

14. **Repeat or Terminate:**

- Repeat the main loop for a new game session or terminate the program based on player choice

Here is how the program works:

The provided C program implements the 15-puzzle game, where a player aims to arrange numbered tiles in ascending order within a 4x4 grid by sliding them into an empty space. The program initializes the grid, displays game rules, and allows the player to make moves using arrow keys. The player has a limited number of moves to achieve the winning state, and the

game checks for valid moves, displaying the outcome when the player either successfully completes the puzzle or exhausts the allowed moves. The code employs functions for matrix manipulation, movement, and rule presentation, using the Windows API for console clearing. Despite its functionality, the code could be improved by addressing security concerns associated with the deprecated gets function and enhancing code structure.

3.) DESCRIPTION OF EACH FUNCTIONALITIES:

Breaking down the main parts of the code and attaching the relevant screen shoots:-

A HEADER FILES:

The inclusion of <windows.h> is for console screen clearing (system("cls")), providing a clean display. <conio.h> is used for non-blocking character input with getch(), allowing the detection of arrow keys for game control, <stdlib.h>declares various utility functions for type conversions, memory allocation, algorithms, and other similar use cases

```
#include<stdio.h>
#include<stdlib.h>
#include<windows.h>
#include<conio.h>
#include<time.h>
```

b.)<u>FIRST FUNCTION:</u> void creatematrix(int a[][4])

This function, creatematrix, generates a 4x4 matrix representing the initial state of the 15-puzzle game. It shuffles the numbers 1 to 15 randomly, ensuring a solvable puzzle configuration, and places them in the array a. The function also initializes the empty space (0) in the matrix. The use of srand(time(0)) seeds the random number generator based on the current time, ensuring a different sequence of shuffled numbers with each program execution.

```
//  function to create a matrix
void creatematrix(int a[][4])
{
  int i,j=1,k,r,ar[16]={0},arr[15];
  srand(time(0));
  for(k=0;k<15;k++)
  {
     r=1+rand()%15;
     if(!ar[r])
        arr[j++]=r;
     else
        k--;
     ar[r]=1;
  }
  for(i=0;i<4;i++)
  {
     for(j=0;j<4;j++)
        a[i][j]=arr[4*i+j+1];
  }
  a[i-1][j-1]=0;
}
```

C.)<u>SECOND FUNCTION</u>: void showmatrix(int a[][4])

The showmatrix function is utilized to display the current state of the 4x4 matrix in the console. It formats and prints the matrix, representing the puzzle's layout, with numbers and an empty space, providing visual feedback to the player during the game.

```c
// function to show 4x4 matrix
void showmatrix(int a[][4])
{
    int i,j;
    printf("\n\n\n----------------------\n");
    for(i=0;i<4;i++)
    {
        printf("| ");
        for(j=0;j<4;j++){
            if(a[i][j]!=0)
                printf("%-2d | ",a[i][j]);
            else
                printf("   | ");
        }
        printf("\n");
    }
    printf("----------------------\n");
}
```

d.) <u>THIRD FUNCTION:</u> int winner(int a[][4])

The winner function checks if the provided 4x4 matrix represents a winning state in the 15-puzzle game. It iterates through the matrix, comparing the numbers to the expected sequence (1 to 15), and returns 1 if the matrix is in the correct order, indicating that the player has won; otherwise, it returns 0.

```c
// function to create winning scenari
int winner(int a[][4])
{
    int i,j,k=1;
    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
            if(a[i][j]!=k&&k!=16)
                break;
        if(j<4)
          break;
    }
    if(j<4)
        return 0;
    else
        return 1;
}
```

e.)FOURTH FUNCTION: void swap(int *p , int *q)

The swap function is used to interchange the values of two integers, pointed to by the pointers p and q. This swapping technique involves arithmetic operations without using a temporary variable. This function is employed in the program to exchange the positions of the empty space (0) and another number in the puzzle matrix, facilitating movement during the 15-puzzle game.

```
//function to swap spaces
void swap(int *p , int *q)
{
    *p=*p+*q;
    *q=*p-*q;
    *p=*p-*q;
    printf("");
}
```

f.) FIFTH FUNCTION : int readEnteredkey()

The readEnteredkey function is used to read a character input from the user, particularly for detecting arrow key inputs during the 15-puzzle game. The function uses getch() to read the character from the console. Since arrow keys typically generate two characters, the function checks if the first character is -32 (an extended code indicating arrow keys on some systems) and reads the second character accordingly. The function

then returns the character representing the user's input. This is essential for capturing user movements in the game.

```
// function to read entered key
int readEnteredkey()
{
    char c;
    c=getch();
    if(c==-32)
        c=getch();
    return c;


return c;
}
```

g.)SIXTH FUNCTION: (Arrow keys functions)

int shiftup(int a[ ][4]) ,
int shiftdown(int a[ ][4]) ,
int shiftleft(int a[ ][4]) ,
int shiftright(int a[ ][4]).

These functions (**shiftup, shiftdown, shiftleft, shiftright**) facilitate to the player's input. They use the swap function to interchange the positions of the empty space and neighboring numbers, ensuring valid moves and updating the game state. The functions return 1 for a successful move and 0 otherwise, helping control the flow of the game based on user input.

```c
// function to shift left numbers
int shiftleft(int a[][4])
{
    int i,j;
    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
            if(a[i][j]==0)
                break;
        if(j<4)
            break;
    }
    if(j==3)
        return 0;
    else{
        swap(&a[i][j],&a[i][j+1]);
        return 1;
    }
}
// function to shift right numbers
int shiftright(int a[][4])
{
    int i,j;
    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
            if(a[i][j]==0)
                break;
        if(j<4)
            break;
    }
    if(j==0)
        return 0;
    else{
        swap(&a[i][j],&a[i][j-1]);
        return 1;
    }
}
```

```c
// function to shift up numbers
int shiftup(int a[][4])
{
    int i,j;
    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
            if(a[i][j]==0)
                break;
        if(j<4)
            break;
    }
    if(i==3)
        return 0;
    else{
        swap(&a[i][j],&a[i+1][j]);
        return 1;
    }
}
// function to shift down numbers
int shiftdown(int a[][4])
{
    int i,j;
    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
            if(a[i][j]==0)
                break;
        if(j<4)
            break;
    }
    if(i==0)
        return 0;
    else{
        swap(&a[i][j],&a[i-1][j]);
        return 1;
    }
}
```

h.)<u>SEVENTH FUNCTION:</u> void gamerule(int a[][4])

The gamerule function is responsible for displaying the rules and instructions of the 15-puzzle game to the user. It provides information on valid moves, the goal of achieving a winning situation, and how to exit the game. The function also shows the initial winning configuration of the puzzle and prompts the user to start the game, enhancing the player's understanding and engagement with the game.

```
//function to show gaming rules
void gamerule(int a[][4])
{
    int i,j,x;
    system("cls");

    printf("              RULE OF THIS GAME\n");
    printf("\n1.You can move only 1 step at a time by arrow key\n");
    printf("Move Up    : By Up arrow key\n");
    printf("Move Down  : By Down arrow key\n");
    printf("Move left  : By Left arrow key\n");
    printf("Move Right : By Right arrow key\n");
    printf("\n2.You can move number at empty position only");
    printf("\n3.For each valid move : your total number of move will decreased by 1");
    printf("\n4.Winning situation : number in a 4*4 matrix should be in order from 1 to 15\n\n");
    printf("              Winning situation\n");
    printf("--------------------\n");
    for(i=0;i<4;i++)
    {
        printf("|");
        for(j=0;j<4;j++){
            if(a[i][j]!=0)
                printf("%2d | ",(4*i+j+1));
            else
                printf("   |");
        }
        printf("\n");
    }
    printf("--------------------\n");
    printf("\n5.You can exit the game at any time by pressing 'E' or 'e'");
    printf("\nSo Try to win in minimum no. of move\n\n");
    printf("          Happy gaming , Good luck\n\n");
    printf("Enter any key to start......");
    x=readEnteredkey();
}
```

j.)<u>EIGHT FUNCTION:</u> (Main function) int main()

The main function orchestrates the execution of the 15-puzzle game. It initializes the game state, including the matrix, player's name, and move limit. The function enters a loop where the player interacts with the game, making moves through arrow keys. It continuously updates the display, checks for a winning or losing condition, and exits the game accordingly, providing a user-friendly and interactive experience.

```
//enter to main function
int main()
{
    system("4");
  int a[4][4],key;
  int maxtry=100; //total moves set 100

  char name[20];
  system("cls");
  printf("Enter Your name:");
  gets(name);
  while(1)
  {
      creatematrix(a); //call create matrix function
      gamerule(a); //call gaming rule function

    while(!winner(a)) // call winning function
    {
        system("cls");
        if(!maxtry)
         break;
        printf("\tWelcome %s , Move Remaining : %d",name,maxtry);
        showmatrix(a); //call show matrix function

        key=readEnteredkey(); //call read enter key function
       switch(key)
       {
       case 69:

       case 101:
         printf("\tThanks for Playing\nHit 'Enter' to exit....");
         key=readEnteredkey();
         exit(0);
       case 72:
         if(shiftup(a)) // call shift up func
         maxtry--;
         break;
       case 80:
         if(shiftdown(a)) // call shift down func
         maxtry--;
         break;
       case 75:
         if(shiftleft(a)) // call shift left func
         maxtry--;
         break;
       case 77:
         if(shiftright(a)) // call shift right func
         maxtry--;
         break;
       default:
         printf("\t\tNot allowed");
       }
    }
    if(!maxtry){
     printf("\n\n\n\t\tYOU LOSE");
     break;
    }
    else{
     printf("\n\n\n\t CONGRATULATION YOU WON!!!!!");
     break;
    }

  }
return 0;
}
// the end
```

OUTPUT SCREEN:

```
             RULE OF THIS GAME

1.You can move only 1 step at a time by arrow key
Move Up    : By Up arrow key
Move Down  : By Down arrow key
Move left  : By Left arrow key
Move Right : By Right arrow key

2.You can move number at empty position only
3.For each valid move : your total number of move will decreased by 1
4.Winning situation : number in a 4*4 matrix should be in order from 1 to 15

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

5.You can exit the game at any time by pressing 'E' or 'e'
So Try to win in minimum no. of move

         Happy gaming , Good luck

Enter any key to start......
```

```
         Welcome Rayyan , Move Remaining : 100


----------------------
| 9   | 8   | 12 | 13 |
| 14  | 6   | 10 | 5  |
| 7   | 11  | 3  | 4  |
| 1   | 2   | 15 |    |
----------------------
         Thanks for Playing
Hit 'Enter' to exit....
--------------------------------
Process exited after 701 seconds with return value 0
Press any key to continue . . .
```

THE PROGRAM THEN ENDS!!

4.) INDIVIDUAL CONTRIBUTIONS :

Most of the work was done in the group but some individual contributions of group members are:

MUHAMMAD RAYYAN LODHI (CTCY-010):

- Giving the idea of the project.
- Mainly design the report of the project
- Breaks the program in separate parts and defines it function in the report. And Attaches the screenshots of relevents code in divided form.
- Created a functions of gaming rules, printing and showing 4x4 matrix.

SYED M SAQIB FAHEEM (CTCY-011):

- Identifies the bugs in the code and debug them.
- Created a functions of read entered key, swap and main function.
- Find out the limitations and future enhancement in game
- Created also a winner function.

SYED M FAIZAN HYDER (CTCY-026):

- Creates the comment for almost each syntaxes.
- Created functions of arrow keys.
- Helps in identify the errors and debug them.
- Created the Pesudo Code of the program.

5.) NEW CONCEPTS THAT LEARNT WHILE WORKING ON THE PROJECT:

We have learnt a lot of new things while working on this project, like we have learnt :

LIMITATIONS:

- Input: getch() might not be available on all systems. Consider using getchar() or cross-platform input libraries
- Input: getch() might not be available on all systems. Consider using getchar() or cross-platform input libraries
- Input validation: Add checks to ensure users enter valid numbers or keys within expected ranges
- Modularity: Break down code into smaller, well-defined functions to improve organization and reusability.
- Error handling: Consider handling potential errors like invalid input or memory allocation failures gracefully.

FUTURE ENHANCEMENT:

- Validate user input to ensure only valid keys and numbers are entered
- Replace gets(name) with fgets(name, 20, stdin) to prevent buffer overflows.
- Cross-platform input libraries or getchar() instead of getch().
- Display a countdown timer for remaining moves to create excitement Offer difficulty levels with varying board sizes or move limits.

- Thorough testing with various inputs and scenarios to identify and fix bugs.
- Implement a hint system to guide players when stuck.Allow customization of game colors and sounds.Create a multi-player mode for competitive play.

CONCLUSION:

The goal is to arrange the numbers 1 to 15 sequentially within a 4x4 matrix, with one empty space for tile movement. **ABOUT GAME** Players move tiles using arrow keys. Valid moves slide a tile into the empty space. Each valid move decreases the allowed moves count (100 maximum). The game ends when the player arranges the tiles in order or runs out of moves.

- **creatematrix:** Generates a random initial matrix with numbers 1 to 15.
- **showmatrix:** Displays the current game matrix.
- **winner**: Checks if the matrix is in winning state.
- **swap:** Swaps the values of two integer pointers (used for tile movement).
- **readEnteredkey:** Reads a key press, handling extended keys.
- **shiftup, shiftdown, shiftleft, shiftright:** Perform tile movements in respective directions.
- **gamerule:** Displays game rules and winning condition.

Overall, the program provides a basic implementation of a sliding puzzle game with clear rules and a user-friendly interface.

THE END