FATIMA JINNAH WOMEN UNIVERSITY

Department of Computer science

Semester: BCS-I(b)

Subject: Programming Fundamentals

Submitted to: Sir Adeel Khalid

Project Title: Tic Tac Toe

Submitted by:

Aneeza Ashraf (fa24b1-cs-096)

Ayesha Mehboob (fa24b1-cs-095)

Zainab Khatoon (fa24b1-cs-019)

Rimsha Ulfat (fa24b1-cs-063)

TABLE OF CONTENT:

TIC TAC TOE:

INTRODUCTION:

The Tic Tac Toe game is a popular two-player game commonly played on paper or through different software platforms. The goal of this project is to create a straightforward yet functional version of Tic Tac Toe in C, utilizing essential programming concepts like arrays, functions (both built-in and user-defined), structures, strings, and file handling. This project serves as a practical way to apply and strengthen the understanding of fundamental C programming principles.

PROJECT OBJECTIVES:

- Develop a working Tic Tac Toe game in C.
- Apply core C programming concepts, including arrays, strings, structures, functions, and file handling.
- Gain an understanding of handling user input and implementing game logic.
- Improve problem-solving and debugging abilities.

SYSTEM REQUIREMENTS:

- **Hardware:** Any computer with a C compiler installed.
- **Software:** C compiler (e.g., GCC), text editor or IDE**.**

KEY CONCEPT USED:

- **Header Files:**

Header files in C define function declarations, constants, and data types. They are included in the main program to enhance modularity and reusability of the code. In this project, standard header files are used to manage input/output operations and string manipulation.

The following header files are utilized:

- **stdio.h:** Provides functions for input/output operations such as **`printf`**, **`scanf`**, and **`fgets`**.
- **stdlib.h**: Offers functions for memory allocation, process control, and various utilities (used minimally in this project).
- **string.h**: Contains functions for string manipulation, such as **`strcspn`**.

- **ARRAY:**

  Arrays are employed to represent the Tic Tac Toe game board. A 2D array (3x3) is used to store the status of each cell, which can be 'X', 'O', or ' ' for an empty cell. This approach simplifies managing the board and checking for a winner.

- **STRING:**

  Strings are utilized to store the names of the players. The **fgets()** function is used to securely capture the player names from the user, while **strcspn()** is applied to remove the newline character at the end of the input.

- **FILE HANDLING:**

  File handling is used to store the game results in a text file **(game_results.txt)**. After each game, the outcome (whether a winner or a draw) is saved along with the players' names.

- **STRUCTURE:**

  Structures are used to combine related data into one unit. In this project, a struct is employed to represent the game state, which contains the **3x3 board** and the current player. This structure allows easier management of the game state, keeping all necessary information in one place.

GAME LOGIC:

The game works like this:

- Players are asked to enter their names.
- The game board is set up and shown.
- Players take turns, choosing a number (1-9) to place their 'X' or 'O' on the board.
- After each move, the program checks if there's a winner or if the game is a draw.
- The game ends when one player wins or the board is full, resulting in a draw.
- The game results are saved in a file and shown at the end.

PROGRAM DESIGN:

The program is divided into the following functions:

- **initializeBoard()**: Initializes the game board by setting all cells to empty spaces.
- **boardDisplay()**: Displays the current state of the game board, showing the positions of 'X', 'O', and empty spaces.
- **move()**: Accepts the player's move (a number between 1-9), checks if the chosen position is valid, and updates the board.
- **checkingWinner()**: Checks if the current player has won the game by examining the rows, columns, and diagonals.
- **boardFull()**: Checks if the board is full, which would result in a draw if no winner is found.
- **switchPlayer()**: Switches the turn to the next player ('X' or 'O').
- **saveResult()**: Saves the results of the game (the winner or draw) to a file named **tic_tac_toe.txt**
- **gameplay()**: Manages the flow of the game by calling the other functions, ensuring the game proceeds turn-by-turn until there is a winner or a draw.

CODE IMPLEMENTATION:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


struct Game {
```

```
    char board[3][3];
    char currentPlayer;
};

void initializeBoard(struct Game *game) {
    for (int row = 0; row < 3; row++) {
        for (int col = 0; col < 3; col++) {
            game->board[row][col] = ' ';
        }
    }
}

void boardDisplay(struct Game game) {
    printf("\nCurrent Game Board:\n");
    for (int row = 0; row < 3; row++) {
        for (int col = 0; col < 3; col++) {
            printf("%c", game.board[row][col]);
            if (col < 2) {
                printf(" | ");
            }
        }
        printf("\n");
        if (row < 2) {
            printf("---------\n");
        }
    }
    printf("\n");
}
```

```
void makeMove(struct Game *game, int move) {
    int row = (move - 1) / 3;
    int col = (move - 1) % 3;
    if (game->board[row][col] == ' ') {
        game->board[row][col] = game->currentPlayer;
    } else {
        printf("Invalid move! The cell is already taken.\n");
    }
}


int checkingWinner(struct Game game) {
    for (int i = 0; i < 3; i++) {
        if (game.board[i][0] == game.board[i][1] && game.board[i][1] == game.board[i][2] &&
game.board[i][0] != ' ') {
            return 1;
        }
        if (game.board[0][i] == game.board[1][i] && game.board[1][i] == game.board[2][i] &&
game.board[0][i] != ' ') {
            return 1;
        }
    }
    if (game.board[0][0] == game.board[1][1] && game.board[1][1] == game.board[2][2] &&
game.board[0][0] != ' ') {
        return 1;
    }
    if (game.board[0][2] == game.board[1][1] && game.board[1][1] == game.board[2][0] &&
game.board[0][2] != ' ') {
        return 1;
    }
    return 0;
}
```

```
int boardFull(struct Game game) {
    for (int row = 0; row < 3; row++) {
        for (int col = 0; col < 3; col++) {
            if (game.board[row][col] == ' ') {
                return 0;
            }
        }
    }
    return 1;
}

void switchPlayer(struct Game *game) {
    if (game->currentPlayer == 'X') {
        game->currentPlayer = 'O';
    } else {
        game->currentPlayer = 'X';
    }
}

void saveResult(const char *player1, const char *player2, char winner) {
    FILE *file = fopen("game_results.txt", "a");
    if (file == NULL) {
        printf("Error opening file to save results.\n");
        return;
    }

    if (winner == 'D') {
        fprintf(file, "Player 1: %s, Player 2: %s, Result: Draw\n", player1, player2);
```

```
    } else {
        fprintf(file, "Player 1: %s, Player 2: %s, Winner: %c\n", player1, player2, winner);
    }
    fclose(file);
}

void gameplay(struct Game *game, const char *player1, const char *player2) {
    int move;
    int winner = 0;

    while (1) {
        boardDisplay(*game);
        printf("Player %c, enter your move (1-9): ", game->currentPlayer);
        scanf("%d", &move);

        if (move < 1 || move > 9) {
            printf("Invalid move! Choose a number between 1 and 9.\n");
            continue;
        }

        makeMove(game, move);
        winner = checkingWinner(*game);

        if (winner) {
            boardDisplay(*game);
            printf("Player %c wins!\n", game->currentPlayer);
            saveResult(player1, player2, game->currentPlayer);
            break;
        }
```

```
        if (boardFull(*game)) {
            boardDisplay(*game);
            printf("It's a draw!\n");
            saveResult(player1, player2, 'D');
            break;
        }

        switchPlayer(game);
    }
}

int main() {
    struct Game game;
    game.currentPlayer = 'X';
    char player1[50], player2[50];
    char playAgain;

    do {
        printf("Enter name of Player 1: ");
        fgets(player1, sizeof(player1), stdin);
        player1[strcspn(player1, "\n")] = '\0';

        printf("Enter name of Player 2: ");
        fgets(player2, sizeof(player2), stdin);
        player2[strcspn(player2, "\n")] = '\0';

        initializeBoard(&game);
        gameplay(&game, player1, player2);
```

```
        printf("\nDo you want to play again? (y/n): ");
        scanf(" %c", &playAgain);
        getchar();

    } while (playAgain == 'y' || playAgain == 'Y');

    printf("Thanks for playing!\n");
    return 0;
}
```

RESULT:

```
Enter name of Player 1: ayesha
Enter name of Player 2: aneeza

Current Game Board:
  |   |
---------
  |   |
---------
  |   |

Player X, enter your move (1-9): 1

Current Game Board:
X |   |
---------
  |   |
---------
  |   |

Player O, enter your move (1-9): 4

Current Game Board:
X |   |
---------
O |   |
---------
  |   |
```

```
Player X, enter your move (1-9): 5

Current Game Board:
X |   |
---------
O | X |
---------
  |   |

Player O, enter your move (1-9): 8

Current Game Board:
X |   |
---------
O | X |
---------
  | O |

Player X, enter your move (1-9): 9

Current Game Board:
X |   |
---------
O | X |
---------
  | O | X

Player X wins!

Do you want to play again? (y/n): n
Thanks for playing!
```