```
/**
--------------------------------------------------------------------------------------
----------------------
* This program allows two people to play the game connect 4. The object of this game
is to place 4 of your pieces in a row in the game board. The 4 pieces can be either
horizontal, vertical, or diagonal. The player gets to choose a column and their token
slides down the column until it hits the bottom or it lands on another piece. If the
player enters U or u the last moves undone and the board returns to the state it was
before the last move. This can continue until the board is empty. Each time U (or u)
is entered the turn changes. If the X player entered U, this undo's O's last move and
it becomes O's turn again. Also, if the player enters P (or p) all the boards that
occurred after each of the moves that have been made are printed.
* @author Safa Patel
--------------------------------------------------------------------------------------
------------------------*/
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
using namespace std;

// Function to display the current game board
// Receives a 2D array representing the game board
// Does not return anything
void displayBoard(char grid[6][7])
{
   // Code for displaying the game board
   cout << "0 1 2 3 4 5 6 " << "\n";
   for (int row = 0; row < 6; row++)
   {
       cout << "\t";
       for (int column = 0; column < 7; column++)
       {
           cout << grid[row][column] << " ";
       }
       cout << endl;
   }
}

// Function to check if a move is valid in a given column
// Receives a column index and a 2D array representing the game board
// Returns a boolean indicating whether the move is valid or not
```

```
bool validMove(int column, char grid[6][7])
{
   // Code for checking if the move is valid
   return column >= 0 && column < 7 && grid[0][column] == '-';
}

// Function to make a move in a given column
// Receives a column index, a player token, and a 2D array representing the game board
// Does not return anything
void makeMove(int column, char player, char grid[6][7])
{
   // Code for making a move in the specified column
   for (int row = 6 - 1; row >= 0; row--)
   {
       if (grid[row][column] == '-')
       {
           grid[row][column] = player;
           return;
       }
   }
}

// Function to check if a player has won the game
// Receives a player token and a 2D array representing the game board
// Returns a boolean indicating whether the player has won or not
bool checkWin(char player, char grid[6][7])
{
   // Check horizontal
   for (int row = 0; row < 6; row++)
   {
       for (int col = 0; col < 7 - 3; col++)
       {
           if (grid[row][col] == player && grid[row][col + 1] == player &&
               grid[row][col + 2] == player && grid[row][col + 3] == player)
           {
               return true;
           }
       }
   }

   // Check vertical
   for (int row = 0; row < 6 - 3; row++)
```

```
    {
        for (int col = 0; col < 7; col++)
        {
            if (grid[row][col] == player && grid[row + 1][col] == player &&
                grid[row + 2][col] == player && grid[row + 3][col] == player)
            {
                return true;
            }
        }
    }

    // Check diagonal (down-right)
    for (int row = 0; row < 6 - 3; row++)
    {
        for (int col = 0; col < 7 - 3; col++)
        {
            if (grid[row][col] == player && grid[row + 1][col + 1] == player &&
                grid[row + 2][col + 2] == player && grid[row + 3][col + 3] == player)
            {
                return true;
            }
        }
    }

    // Check diagonal (down-left)
    for (int row = 0; row < 6 - 3; row++)
    {
        for (int col = 3; col < 7; col++)
        {
            if (grid[row][col] == player && grid[row + 1][col - 1] == player &&
                grid[row + 2][col - 2] == player && grid[row + 3][col - 3] == player)
            {
                return true;
            }
        }
    }
    return false;
}

// Function to check if the game board is full
// Receives a 2D array representing the game board
// Returns a boolean indicating whether the board is full or not
```

```
bool boardFull(char grid[6][7])
{
   // Code for checking if the board is full
   for (int col = 0; col < 7; col++)
   {
       if (grid[0][col] == '-')
       {
           return false;
       }
   }
   return true;
}

// Function to check if a column is full
// Receives a column index and a 2D array representing the game board
// Returns a boolean indicating whether the column is full or no
bool columnFull(int column, char grid[6][7])
{
   // Code for checking if the column is ful
   if (grid[0][column] == '-')
   {
       return false;
   }
   return true;
}

// Define Node structure
class Node
{
   public:
       Node* next;
       char grid[6][7];

       Node(char currentGrid[6][7], bool printBoard = true)
       {
           for (int i = 0; i < 6; i++)
           {
               for (int j = 0; j < 7; j++)
               {
                   grid[i][j] = currentGrid[i][j];
               }
           }
           // Print the board only if printBoard is true
           if (printBoard) {
               displayBoard(grid);
           }

           next = nullptr;
       }
};

// Define UndoLinkedList
class UndoLinkedList
{
   public:
       Node* head;

       UndoLinkedList(char initialGrid[6][7])
       {
           head = new Node(initialGrid);
           for (int i = 0; i < 6; i++)
           {
               for (int j = 0; j < 7; j++)
               {
                   head->grid[i][j] = initialGrid[i][j];
               }
           }
       }

       // Function to delete the latest node in the linked list
       void deleteNode()
       {
           if(head->next == nullptr)
           {
               return;
           }
           Node* temp = head;
           head = head->next;
           delete temp;
       }

       // Function to add a new node to the linked list
       void addNode()
```

```
        {
            Node* newNodePointer = new Node(head->grid, false);
            for(int i = 0; i < 6; i++)
            {
                for(int j = 0; j < 7; j++)
                {
                    newNodePointer->grid[i][j] = head->grid[i][j];
                }
            }
            newNodePointer-> next = head;
            head = newNodePointer;
        }

        // Function to print the current state of the board
        void printCurrentBoard()
        {
            displayBoard(head->grid);
        }

        // Function to print all moves in the linked list
        void printAllMoves()
        {
            Node* current = head;
            vector<Node*> moves;

            while (current != nullptr)
            {
                moves.push_back(current);
                current = current->next;
            }

            // Print moves in reverse order
            for (int i = moves.size() - 1; i >= 0; i--)
            {
                displayBoard(moves[i]->grid);
            }
        }
};

// Main function for the Connect 4 game
// Manages the overall game logic, including setup, user input, and game state updates
// Does not receive any parameters
// Returns an integer indicating the program's exit status
int main()
{
   // Introduction and game setup
   cout << "This is the Game Connect 4." << endl;
   cout << "Each player should place an X or an O in the space " << endl;
   cout << "by entering the column you want to place the piece." << endl;
   cout << "The piece will fall until it reaches the bottom or " << endl;
   cout << "the current pieces in the board. When X or O gets 4 in " << endl;
   cout << "a row (either horizontally, vertically, or diagonally, "  << endl;
   cout << "then that person wins. The user can enter Q (or q) to " << endl;
   cout << "end the game early." << endl;
   cout << "Let's get started!!!" << endl;

   // Initialize the game board with empty spaces
   char grid[6][7];
   for (int row = 0; row < 6; row++)
   {
       for (int col = 0; col < 7; col++)
       {
           grid[row][col] = '-';
       }
   }

   // Set current player to 'X' and initialize game over to false
   char currentPlayer = 'X';
   bool gameover = false;

   // Instantiation of the UndoLinkedList object, undoList
   UndoLinkedList undoList(grid);

   // Main game loop
   while (!gameover)
   {
       // Take in user move for which column they want to put their piece
       int column;
       cout << "It is " << currentPlayer << "'s turn." << endl;
       cout << "Enter a column to place your piece.";
       string input;
       cin >> input;
       cout << endl;
```

```
        // Check if the player wants to quit the game or undo a move or print boards in
the linked list
        if (input == "Q" || input == "q")
        {
            cout << "Ending Game" << endl;
            break;
        } else if(input == "U" || input == "u")
        {
            if (undoList.head->next != nullptr)
            {
                undoList.deleteNode();
                displayBoard(undoList.head->grid);
                currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
            }
            else
            {
                undoList.deleteNode();
                displayBoard(undoList.head->grid);
            }
        }
        else if (input == "P" || input == "p")
        {
            undoList.printAllMoves();
        }
        else
        {

            // Convert user input of a string number to an int
            column = stoi(input);

            // Check if the chosen column is already full
            if (columnFull(column, grid))
            {
                cout << "column chosen is already full" << endl;
            }

            // Manages the game's turn-based logic, makes sure the moves are valid, and
checks for a win or a draw, ultimately updating the game state
            if (column >= 0 && column <= 6 && validMove(column, grid))
            {
                undoList.addNode();
                makeMove(column, currentPlayer, undoList.head->grid);
                displayBoard(undoList.head->grid);

                if (checkWin(currentPlayer, grid))
                {
                    displayBoard(undoList.head->grid);
                    cout << endl;
                    cout << "Game is Over, Player " << currentPlayer << " got 4 in a row!!!! " << endl;
                    gameover = true;
                }
                else if (boardFull(grid))
                {
                    displayBoard(undoList.head->grid);
                    cout << "Board is Full, It's a Draw!!!" << endl;
                    gameover = true;
                    break;
                }
                else
                {
                    currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
                }
            }
            else
            {
                cout << "Please enter a valid column" << endl;
            }
        }
    }
    return 0;
}
```