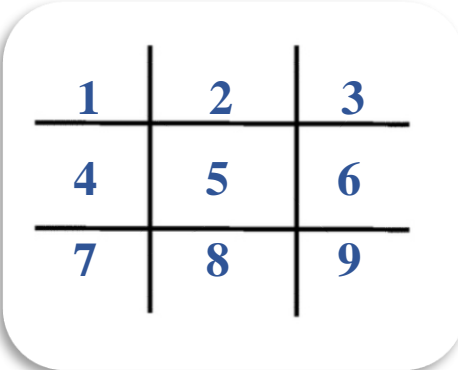


## Game Description and implementation :

Data structures is widely used in a variety of applications and games, they are essential to programming.

We implement the following Tic-Tac-Toe game using linked lists and arrays **FROM SCRATCH.**

We have made linked list with 9 nodes; each node has char type called 'Data' and a pointer pointing to next element called 'Next'. The formed linked list having Data {1,2,3,4,5,6,7,8,9} forming the 3x3 board as shown in Figure 1



1	2	3
4	5	6
7	8	9

*Figure 1*

- So, by making those linked lists we call the function called we now if we want to put 'X' or 'O' at any given index , we call the function

Called : `void insert_at_position(node** head, char data, int pos)`

Which we give it head parameter , data parameter {1,2,3 ...etc. } ,and the position at which we want to insert the required data.

- In order to put 'X' or 'O' at any place in board we implemented the function

Called : `bool change_value(node* head, char data, int index)`

Which we give it the head, data, index at which we want to put on, this function returns True if index is free and it returns False if index has already either 'X' or 'O' .

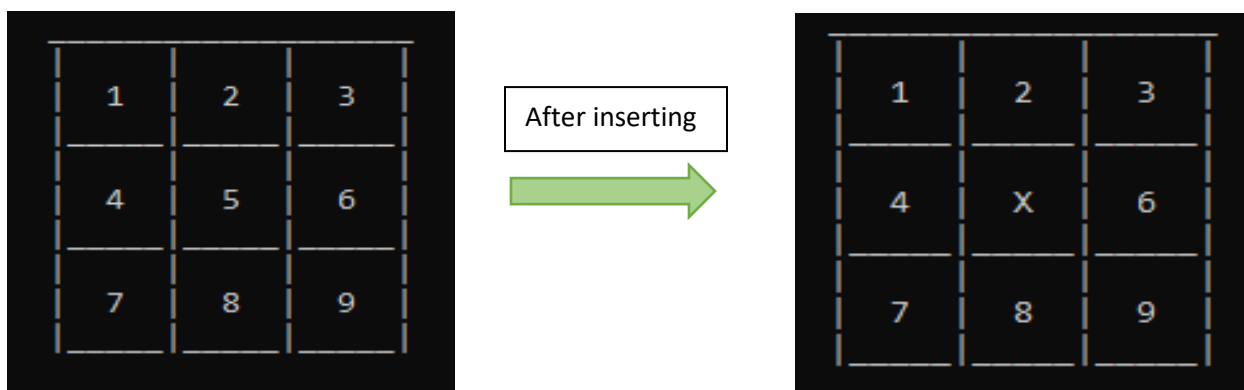
So, for illustration if we want to insert 'X' at position 5 we call the function as shown in Figure 2.

1	2	3
4	X	6
7	8	9

*Figure 2*

- ✚ Example of Real written code of inserting 'X' at position 5 in board , just call the function :

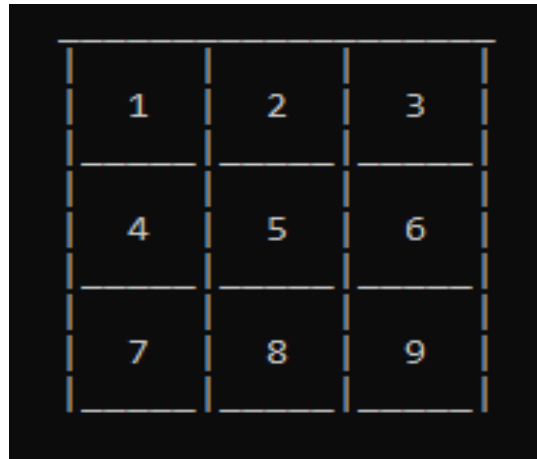
`change_value(head, 'X', 5);`



- To Display the board each time after insertion, we use the function

Called : `display(node* head)`

And give it head only . It displays the following board shown in Figure 3.



1	2	3
4	5	6
7	8	9

*Figure 3*

- To check if either 'X' or 'O' won, or draw happened we implemented the function called : `void iscompleted(node* head)`

Which we give it the head of linked list only, and it checks if any row or column or any diagonal is matched with same Data or neither and in that case, it is a draw.

And we call this function every time after inserting 'X' or 'O' .

Here are a few examples of all situations ( Win 'X' , Win 'O' , Draw)

1	X	O
X	X	X
O	8	O

-----X Won-----

(2<sup>nd</sup> row matched 'X')

O	2	X
O	X	6
O	8	X

-----O Won-----

(1<sup>st</sup> column matched 'O')

O	X	X
4	O	X
7	8	O

-----O Won-----

(Left diagonal matched 'X')

O	O	X
X	X	O
O	X	X

-----Draw-----

(No matching Found. i.e... Draw)

## Types Of Modes:

We implemented this game having 2 modes , either you want to play “Single-Player” (against computer), also it has difficulties -Easy or -Hard .

Or you want to play “Multi-Player”.

### 1) Multi- Player

In this mode 2 users play against each other.

### 2) Single- Player

#### ➤ Easy Difficulty

Here we implemented the function called : `int Easy_difficulty(node* head)`

Which only takes the head of linked list, and this function generates a random number between [1,9] and returns this random number then all we do next is to call `change_value()` function as pass that random number as index.

And since it is a random number then possibility of winning is big, so that's why it is Easy Difficulty.

🎲 Example we put 'X' only in index 5

1	2	0
4	X	6
7	8	9

*Figure 4*

As shown in Figure 4 , if we put 'X' in index 5 , it is noticed that 'O' is put automatically at index 3.

## ➤ Hard Difficulty

Here we implemented the minimax algorithm to generate all winning possible solutions to avoid the user to win

Using the implemented functions called :

```
void draw(int a[9])
```

```
int win(const int board[9])
```

```
int minimax(int board[9], int player)
```

```
void computerMove(int board[9])
```

```
void playerMove(int board[9])
```

✚ Example of a game in hard difficulty

1	2	3
4	5	6
7	8	9

Please enter a number from 1 to 9: 5



1	2	3
0		
4	5	6
	X	
7	8	9



1	2	3
0		X
4	5	6
	X	
7	8	9
0		



1	2	3
0		
4	5	6
	X	
7	8	9

Please enter a number from 1 to 9: 3

1	2	3
0		X
4	5	6
	X	
7	8	9
0		

Please enter a number from 1 to 9: 4



1	2	3
0		X
4	5	6
X	X	0
7	8	9
0		



1	2	3
0		X
4	5	6
X	X	0
7	8	9
0		

Please enter a number from 1 to 9: 2



1	2	3
0	X	X
4	5	6
X	X	0
7	8	9
0	0	



1	2	3
0	X	X
4	5	6
X	X	0
7	8	9
0	0	

Please enter a number from 1 to 9: 9



1	2	3
0	X	X
4	5	6
X	X	0
7	8	9
0	0	X

A draw. How droll.

I played many times and I never Won 😊.

Now it's your turn to try and play to Win.

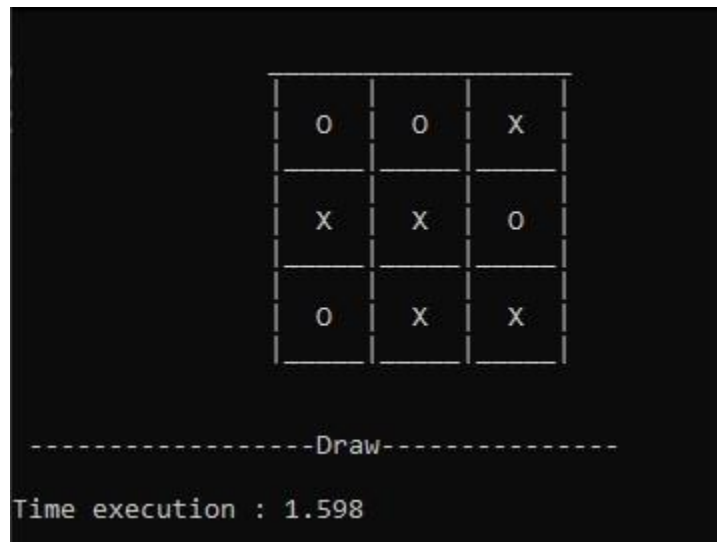
## Time Execution and Analysis:

We calculated the time of our program by making a time counter to get initial time of program and another time counter to get finish time of our program then subtract them to calculate execution time.

### **Note:**

- The calculated time has within it the time at which user takes to type the required index, because the game must require at least a single user to type the index, so we put all required indices all at once and pressed enter, then we observed the time execution

### Time Execution **Before** making Enhancements:



So, time execution = 1.598 sec



### Time Execution **After** making Enhancements:

We noticed that the most complex functions we implemented were the isCompleted() function that checks either a matched row, column or neither,

And the other function was minimax() function, but it was very very complex to enhance it, so we enhanced the isCompleted() function and other functions that were easy as well.

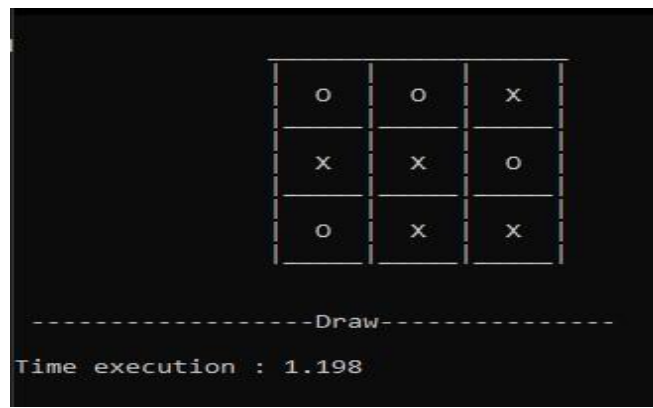
We enhanced the isCompleted() function by getting rid of unnecessary loops and bits of code like :

```
// for (int j = 0; j < 4; j++)  
    tmphead = tmphead->next->next->next->next;  
  
    if (tmphead->data == 'X')  
        Xcounter++;  
  
//for (int j = 0; j < 4; j++)  
    tmphead = tmphead->next->next->next->next;  
    if (tmphead->data == 'X')  
        Xcounter++;
```

The green lines of code were written before enhancement , so we got rid of 2 for loops with iterations.

We have done similar things like that in the entire function so ended up by saving almost **12** for loops with iteration . **That is A Lot ...**

### New Execution Time After Enhancement :



So, it is clearly that execution time decreased significantly than before enhancement.

**New Execution time = 1.198 sec** (old was = 1.598 sec)