SIX WEEKS SUMMER TRAINING REPORT

On

DATA STRUCTURE AND ALGORITHM (SELF PACED)

Submitted by

MORRIGADUDHULA ABHINAY

REGESTRATION NUMBER : 12020232

PROGRAMME NAME : Btech. CSE

Under the Guidance of

Mr. Sandeep Jain

School of Computer Science & Engineering

Lovely Professional University, Phagwara

(June-July,2022)

DECLARATION

I hereby declare that I have completed my six weeks summer training at Geeks for Geeks platform from 1st june 2022 to 15th July 2022 under the guidance of MR. Sandeep Jain. I have declare that I have worked full dedication during there 8 weeks of training and my learning outcomes fulfill the requirements of training for the award of degree of B.tech. CSE , Lovely Proffesional University, Phagwara.

Date – 18th July 2022 Name of Student – M.Abhinay

Registration no: 12020232

ACKNOWLEDGEMENT

I MORRIGADUDHULA ABHINAY, 12020232, hereby declare that the work done by me on “DSA” from June 2022 to July 2022, is a record of original work for the partial fulfillment of the requirements for the award of the degree, Bachelor of Technology.I would like to express my gratitude towards my University as well as Geeks for Geeks for providing me the golden opportunity to do this wonderful summer training regarding DSA, which also helped me in doing a lot of homework and learning. As a result, I came to know about so many new things. So, I am really thank full to them. Moreover I would like to thank my friends who helped me a lot whenever I got stuck in some problem related to my course. I am really thankfull to have such a good support of them as they always have my back whenever I need. Also,I would like to mention the support system and consideration of my parents who have always been there in my life to make me choose right thing and oppose the wrong. I have taken much efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them

Morrigadudhula Abhinay

12020232

INTRODUCTION

DSA self paced course is a complete package that helped me to learn Data Structures and Algorithms from Basic to an Advance level. The course curriculm has been divided into 10 weeks, where I practiced questions and I have attempted the assesment tests accordingly. Data Structure is a way of collecting and organizing data in such a way that we can perform operations on these data in an effective way. Data Structures is about rendering data elements in terms of some relationship, for better organization and storage. An algorithm is a finite set of instructions or logic, written in order, to accomplish a certain

predefined task. Algorithm is not the complete code or program, it is just the core logic(solution) of a problem, which can be expressed either as an informal high-level description as pseudocode or using a flowchart. This course is a complete package that helped me learn Data Structures and Algorithms from basic to an advanced level. The course curriculum has been divided into 8 weeks where one can practice questions & attempt the assessment tests according to his own pace. The course was Self placed means I could join the course anytime and all the content will be avilable to me once I get enrolled. There was video lectures to learn form and multiple choice questions to practice. I learned Algorithmic techniques for solving various problems with full flexibility of time as I was not time bounded. This course does not require any prior knowledge of Data Structure and Algorithms, but a basic knowledge of any programming language (C++ / Java) will be helpful, it helps us to increase our problem solving skill.

WEEKLY OVERVIEW OF INTERNSHIP ACTIVITIES

Week 1:

Day 1: Introduction- Analysis of Algorithms, Asymptotic Notation, Time Complexity, Space Complexity.

Day 2,3,4 : • Finding the number of digits in a number.
- Arithmetic and Geometric Progressions.
- Quadratic Equations.
- Mean and Median.
- Prime Numbers.
- LCM and HCF

- Factorials
- Permutations and Combinations
- Modular Arithmetic

Day 5,6: Bit Manipulation- Bitwise operators
Set Bits, Count Bits, Power of 2 and Power sets, One odd occurring, 2 odd occurring, Practice Questions.

- **Day 7:** Recursion- Application, Tail Recursion, Base Conditions.Classic Recursion Problems including Tower of Hanoi and Josephus Problem, Practice Questions.

Week 2:

Day 1: Continuation of Week 1 Day 7 topics.

DAY 2,3,4 :

- Arrays- Introduction and simple problemsShifting and Rotation.
- Stock buy and sell, Trapping Rainwater.
- Kadane's Algo, Prefix Sum, Sliding window, Practice Questions.

Day 5,6:

- Searching- Linear, Binary, Ternary Search, Modified Binary Search to find floor, ceil.
- index of 1 st and last occurrence, 2 pointers, median in sorted array
- Find Square/Cube root, Search in rotated Sorted Array.
- Allocate Minimum pages, Practice Questions.
- **Day 7:** Sorting- All types of sorting with best, avg, worst time complexity,
- Problems using merge sort and Quick Sort implementation
- Comparator Function in Cpp/Java inbuilt sort function
- Practice Questions.

Week 3:

Day 1,2: Continuation of Week 2 Day 7 topics.

Day 3,4: Matrix- Terminologies and Properties

- Transpose, Rotation, Spiral, Multiplication of matrices
- Search in Sorted matrix
- Median in row sorted Matrix
- Practice Questions.

Day 5,6:

Hashing- Concept of hashing

- Separate chaining and open addressing,
- implementation of hashing in Cpp/Java, Practice Questions.

Day 7: Strings- Introduction and basic questions, String Matching Algorithms, Lexicographic rank of string, Longest substring with distant characters, Practice Questions.

Week 4:

Day 1,2: Continuation of Week 3 Day 7 topics.

- **Day 3,4,5:** Linked List- Types (Single LL, Double LL, XOR LL, Circular LL), Advantages, implementation and operations.
- Practice questions.

Day 6,7: Stacks- Introduction,

- implementation and operations using array and linked list
- Practice Questions.

Week 5:

Day 1: Continuation of Week 4 Day 7 topics.

- **Day 2,3:** Queue and Deque- - Introduction
- implementation and operations using array and linked list, few Questions
- implement stack using queues and vice versa, Practice Questions.

Day 4,5,6: Trees- Introduction and types of trees

- Creation, Deletion, Insertion in a tree, Binary trees
- Practice Questions.

Day 7: Binary Search Trees- Trees- Introduction and properties of BST, Creation, Searching, Deletion, Insertion in a BST, implementation of BST and Practice Questions.

Week 6:

Day 1,: Continuation of Week 5 Day 7 topics.

Day 2,3: Heaps- Introduction to Heap, Creation, Insertion, Deletion, Heapify, Priority Queues and related application in questions. Practice Questions.

Day 4,5,6,7: Graphs- Introduction to Graphs, properties, and representations, traverse the graphs using BFS and DFS, detect cycles, find shortest paths, find strongly connected components, etc. in a graph. Practice Questions on BFS, DFS and all given algorithms.

Week 7:

Day 1,2: Greedy - Practice Questions related to different Greedy approaches given.

Day 3,4: Backtracking- Introduction, Rat in a maze, N Queen Problem, Suduko Problem. Practice questions.

Day 5,6,7: Dynamic Programming- Introduction to Dynamic Programming

- overlapping subproblems, and optimal substructures,
- Top-down and bottom-up approaches to solving a DP problem,
- All different types of DP problems and their variations.
- Practice Questions.

Week 8:

Day 1: Continuation of Week 7 Day 7 topics.

Day 2,3: Tries- Introduction to Tries, insert, search, delete, and updating in Tries, Practice Questions.

Day 4,5: Segment Trees (Basic Level)- Introduction to Segment Trees and when to use them, applying segment-trees in questions, Practice Questions.

- **Day 6,7:** Disjoint Set- Introduction to DSU,
- Union by Rank and Path Compression
- Kruskal's algorithm, Practice Questions

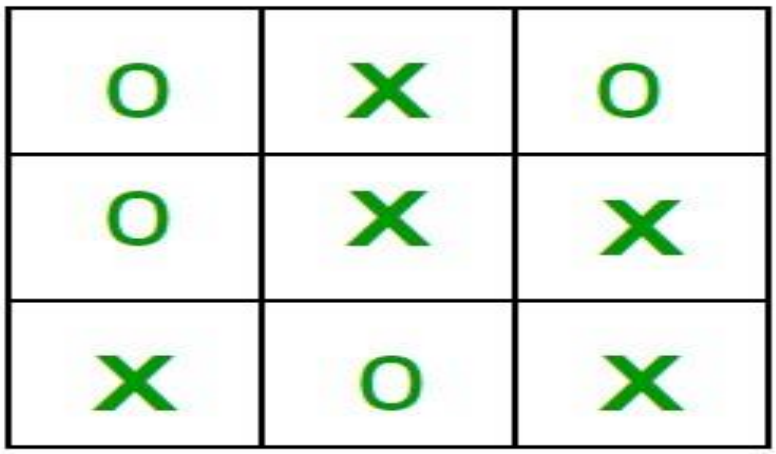
PROJECT DESCRIPTION

I have choosen Tic Tac Toe as my project

Implementation of Tic-Tac-Toe game

Rules of the Game:

- The game is to be played between two people (in this program between HUMAN and COMPUTER).
- One of the player chooses ‘O’ and the other ‘X’ to mark their respective cells.
- The game starts with one of the players and the game ends when one of the players has one whole row/ column/ diagonal filled with his/her respective character (‘O’ or ‘X’).
- If no one wins, then the game is said to be draw.



Implementation :

In our program the moves taken by the computer and the human are chosen randomly. We use rand() function for this.

What more can be done in the program?

The program is in not played optimally by both sides because the moves are chosen randomly. The program can be easily modified so that both players play optimally

(which will fall under the category of Artificial Intelligence). Also the program can be modified such that the user himself gives the input (using scanf() or cin).

The above changes are left as an exercise to the readers.

Winning Strategy – An Interesting Fact

If both the players play optimally then it is destined that you will never lose ("although the match can still be drawn"). It doesn't matter whether you play first or second.In another ways – " Two expert players will always

draw ".

Isn't this interesting ?

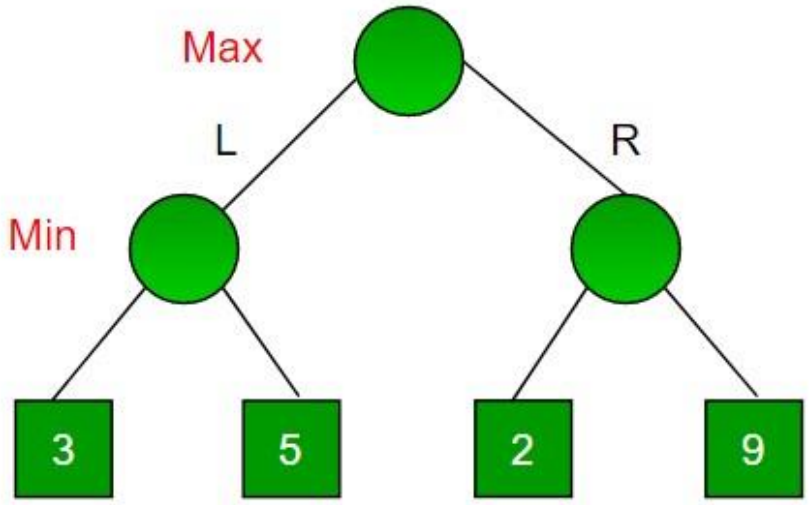
I used minmax algorithm in this project.

Minimax Algorithm :

Minimax is a kind of backtracing algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally. It is widely used in two player turn-based games such as Tic-Tac-Toe, Backgammon, Mancala, Chess, etc.
In Minimax the two players are called maximizer and minimizer. The **maximizer** tries to get the highest score possible while the **minimizer** tries to do the opposite and get the lowest score possible.

Every board state has a value associated with it. In a given state if the maximizer has upper hand then, the score of the board will tend to be some positive value. If the minimizer has the upper hand in that board state then it will tend to be some negative value. The values of the board are calculated by some heuristics which are unique for every type of game.

Example:
Consider a game which has 4 final states and paths to reach final state are from root to 4 leaves of a perfect binary tree as shown below. Assume you are the maximizing player and you get the first chance to move, i.e., you are at the root and your opponent at next level.

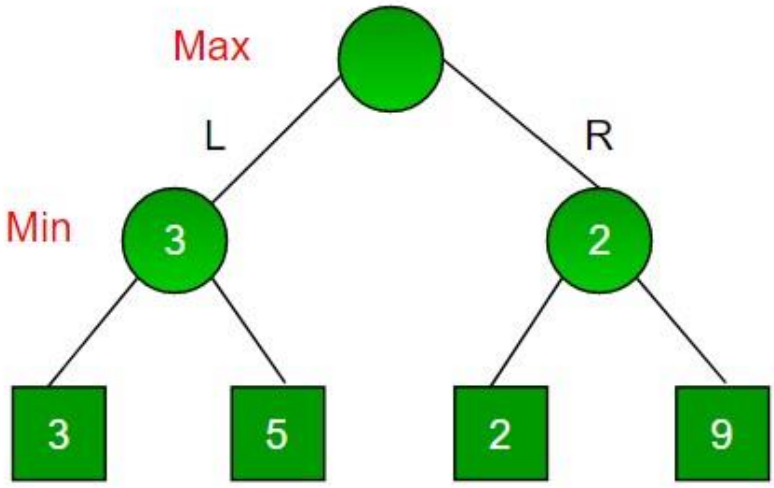


Since this is a backtracking based algorithm, it tries all possible moves, then backtracks and makes a decision.

- Maximizer goes LEFT: It is now the minimizers turn. The minimizer now has a choice between 3 and 5. Being the minimizer it will definitely choose the least among both, that is 3
- Maximizer goes RIGHT: It is now the minimizers turn. The minimizer now has a choice between 2 and 9. He will choose 2 as it is the least among the two values.

Being the maximizer you would choose the larger value that is 3. Hence the optimal move for the maximizer is to go LEFT and the optimal value is 3.

Now the game tree looks like below :



The above tree shows two possible scores when maximizer makes left and right moves.

CODING :

```
// A C++ Program to play tic-tac-toe

#include<bits/stdc++.h> using
namespace std;

#define COMPUTER 1
#define HUMAN 2

#define SIDE 3 // Length of the board
```

```
// Computer will move with 'O'
// and human with 'X'
#define COMPUTERMOVE 'O'
#define HUMANMOVE 'X'

// A function to show the current board status void
showBoard(char board[][SIDE])
{
printf("\n\n");

  printf("\t\t\t %c | %c | %c \n", board[0][0],
board[0][1], board[0][2]);   printf("\t\t\t-------
-------\n");   printf("\t\t\t %c | %c | %c \n",
board[1][0],              board[1][1],
board[1][2]);   printf("\t\t\t--------------\n");
printf("\t\t\t %c | %c | %c \n\n", board[2][0],
board[2][1], board[2][2]);

  return;
}

// A function to show the instructions void
showInstructions()
{   printf("\t\t\t Tic-Tac-Toe\n\n");   printf("Choose a
cell numbered from 1 to 9 as below"
      " and play\n\n");
```

```
  printf("\t\t\t 1 | 2 | 3 \n");   printf("\t\t\t--
------------\n");   printf("\t\t\t 4 | 5 | 6 \n");
printf("\t\t\t--------------\n");   printf("\t\t\t
7 | 8 | 9 \n\n");

  printf("-\t-\t-\t-\t-\t-\t-\t-\t-\n\n");

  return; }


// A function to initialise the game void
initialise(char board[][SIDE], int moves[])
{
  // Initiate the random number generator so that
// the same configuration doesn't arises
srand(time(NULL));

  // Initially the board is empty
for (int i=0; i<SIDE; i++)
  {     for (int j=0; j<SIDE;
j++)       board[i][j] = ' ';
  }

  // Fill the moves with numbers
for (int i=0; i<SIDE*SIDE; i++)
moves[i] = i;
```

```
  // randomise the moves
random_shuffle(moves, moves + SIDE*SIDE);

  return; }

// A function to declare the winner of the game void
declareWinner(int whoseTurn)
{
  if (whoseTurn == COMPUTER)
printf("COMPUTER has won\n");
  else
    printf("HUMAN has won\n");
return; }

// A function that returns true if any of the row
// is crossed with the same player's move bool
rowCrossed(char board[][SIDE])
{   for (int i=0; i<SIDE;
i++)
  {     if (board[i][0] == board[i][1]
&&       board[i][1] == board[i][2]
&&       board[i][0] != ' ')       return
(true);
  }
return(false);
}
```

```
// A function that returns true if any of the column
// is crossed with the same player's move bool
columnCrossed(char board[][SIDE])
{   for (int i=0; i<SIDE;
i++)
  {      if (board[0][i] == board[1][i]
&&         board[1][i] == board[2][i]
&&       board[0][i] != ' ')       return
(true);
  }
return(false);
}

// A function that returns true if any of the diagonal
// is crossed with the same player's move bool
diagonalCrossed(char board[][SIDE])
{   if (board[0][0] == board[1][1]
&&     board[1][1] == board[2][2]
&&     board[0][0] != ' ')
return(true);

  if (board[0][2] == board[1][1] &&
board[1][1] == board[2][0] &&
board[0][2] != ' ')     return(true);

  return(false);
}
```

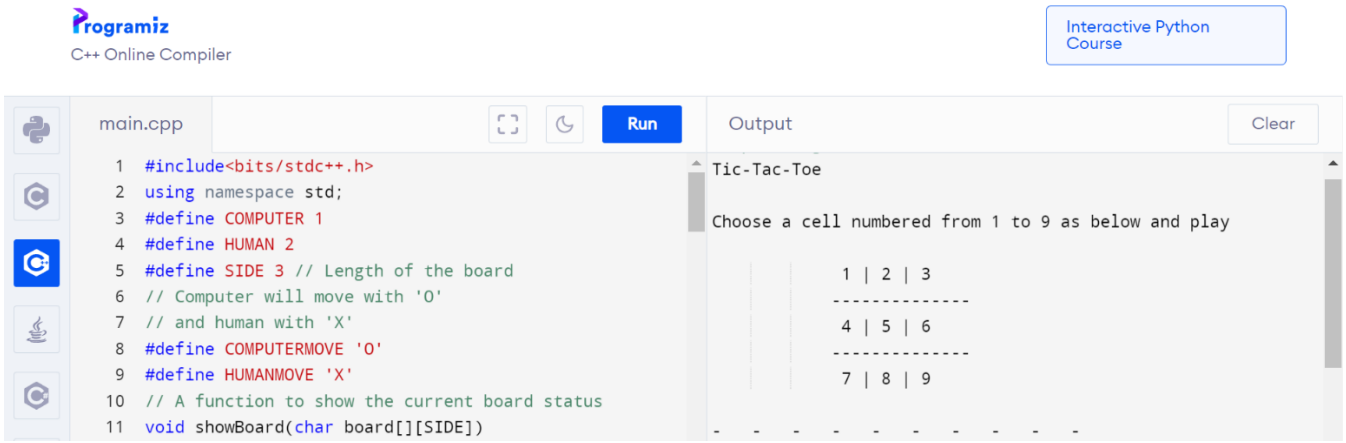
```
// A function that returns true if the game is over
// else it returns a false
bool gameOver(char board[][SIDE])
{   return(rowCrossed(board) ||
columnCrossed(board)
     || diagonalCrossed(board) );
}

// A function to play Tic-Tac-Toe void
playTicTacToe(int whoseTurn)
{
  // A 3*3 Tic-Tac-Toe board for playing
char board[SIDE][SIDE];

  int moves[SIDE*SIDE];

  // Initialise the game
initialise(board, moves);

  // Show the instructions before playing
showInstructions();

  int moveIndex = 0, x, y;

  // Keep playing till the game is over or it is a draw
while (gameOver(board) == false &&
moveIndex != SIDE*SIDE)
```

```
  {
    if (whoseTurn == COMPUTER)
    {
      x = moves[moveIndex] / SIDE;       y =
moves[moveIndex] % SIDE;       board[x][y] =
COMPUTERMOVE;       printf("COMPUTER has
put a %c in cell %d\n",          COMPUTERMOVE,
moves[moveIndex]+1);       showBoard(board);
moveIndex ++;       whoseTurn = HUMAN;
    }

    else if (whoseTurn == HUMAN)
    {
x = moves[moveIndex] / SIDE;
y = moves[moveIndex] % SIDE;       board[x][y] =
   HUMANMOVE;       printf ("HUMAN has put a %c in cell
   %d\n",          HUMANMOVE, moves[moveIndex]+1);
   showBoard(board);       moveIndex ++;       whoseTurn =
   COMPUTER;
    }
  }

  // If the game has drawn   if
(gameOver(board) == false &&
moveIndex == SIDE * SIDE)
printf("It's a draw\n");   else
  {
```

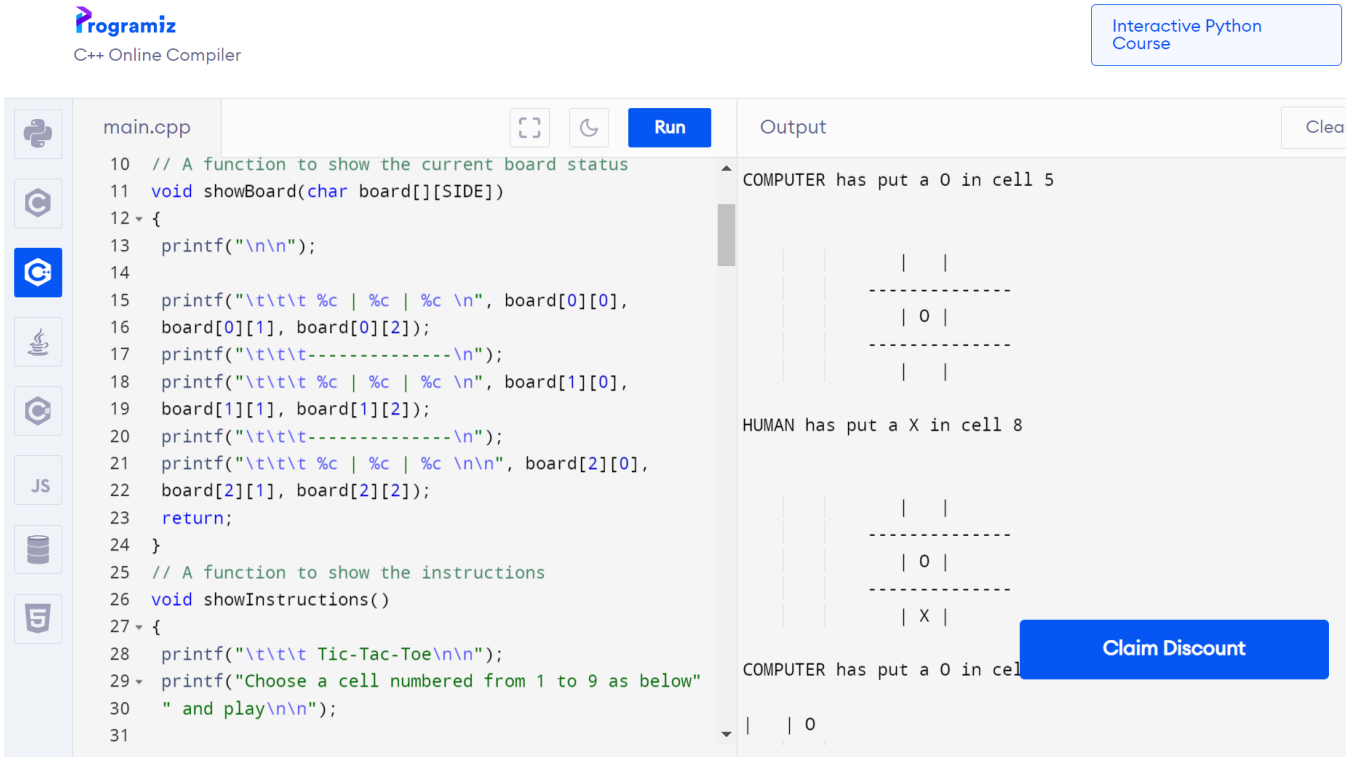
```
    // Toggling the user to declare the actual
    // winner
    if (whoseTurn == COMPUTER)
whoseTurn = HUMAN;     else if
(whoseTurn == HUMAN)
whoseTurn = COMPUTER;

    // Declare the winner
declareWinner(whoseTurn);
  }
return; }


// Driver program int
main()
{
  // Let us play the game with COMPUTER starting first
playTicTacToe(COMPUTER);

  return (0);
}
```

SCREENSHOTS:

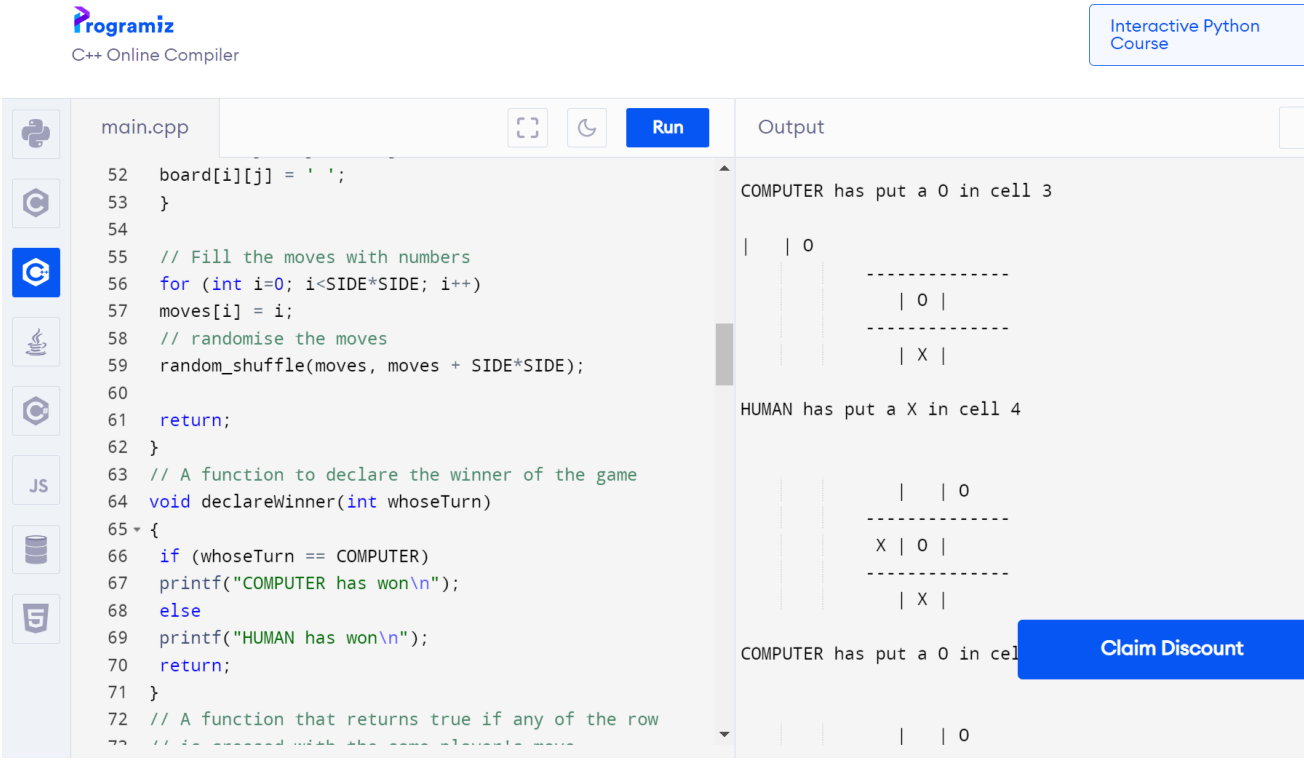
- First it will ask to who else want to start the game if human want to start the game type yes or else no



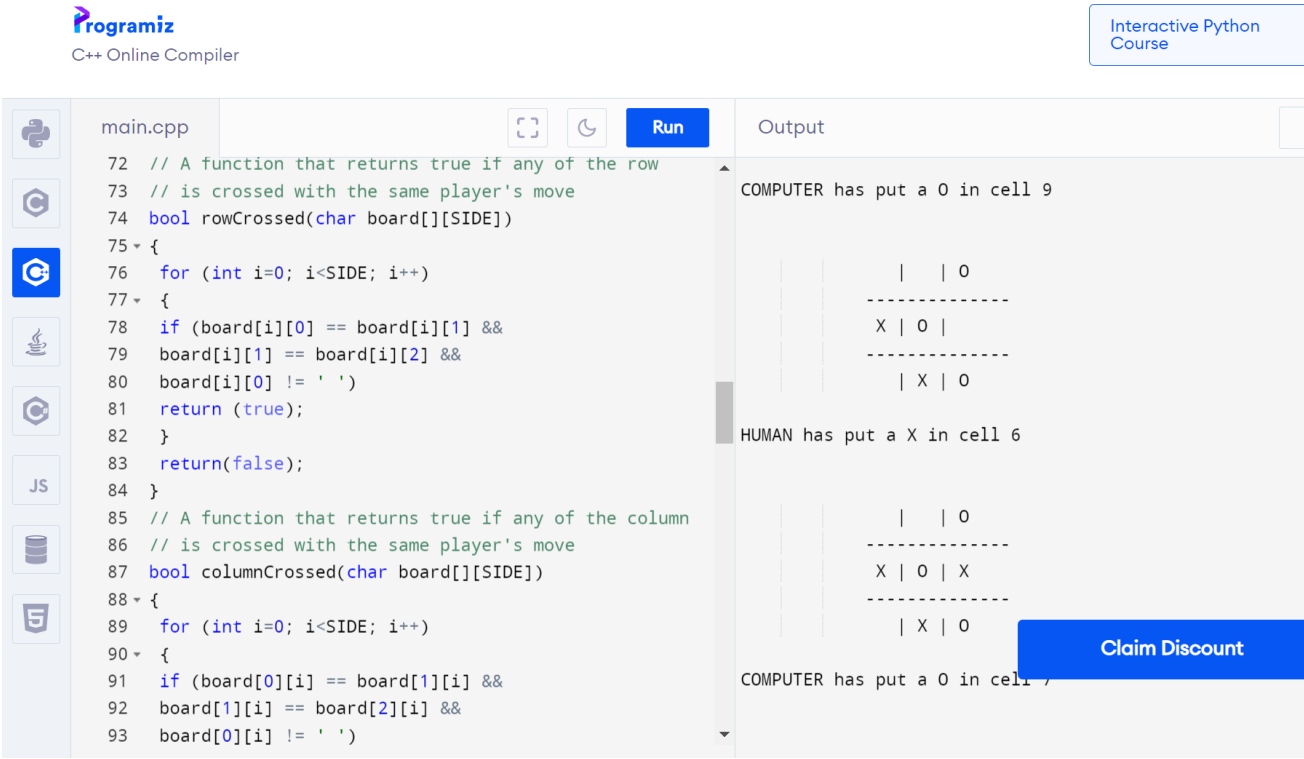
❖ Computer was 1st started the game and computer has chosen 5th position and human has chosen 8th position.



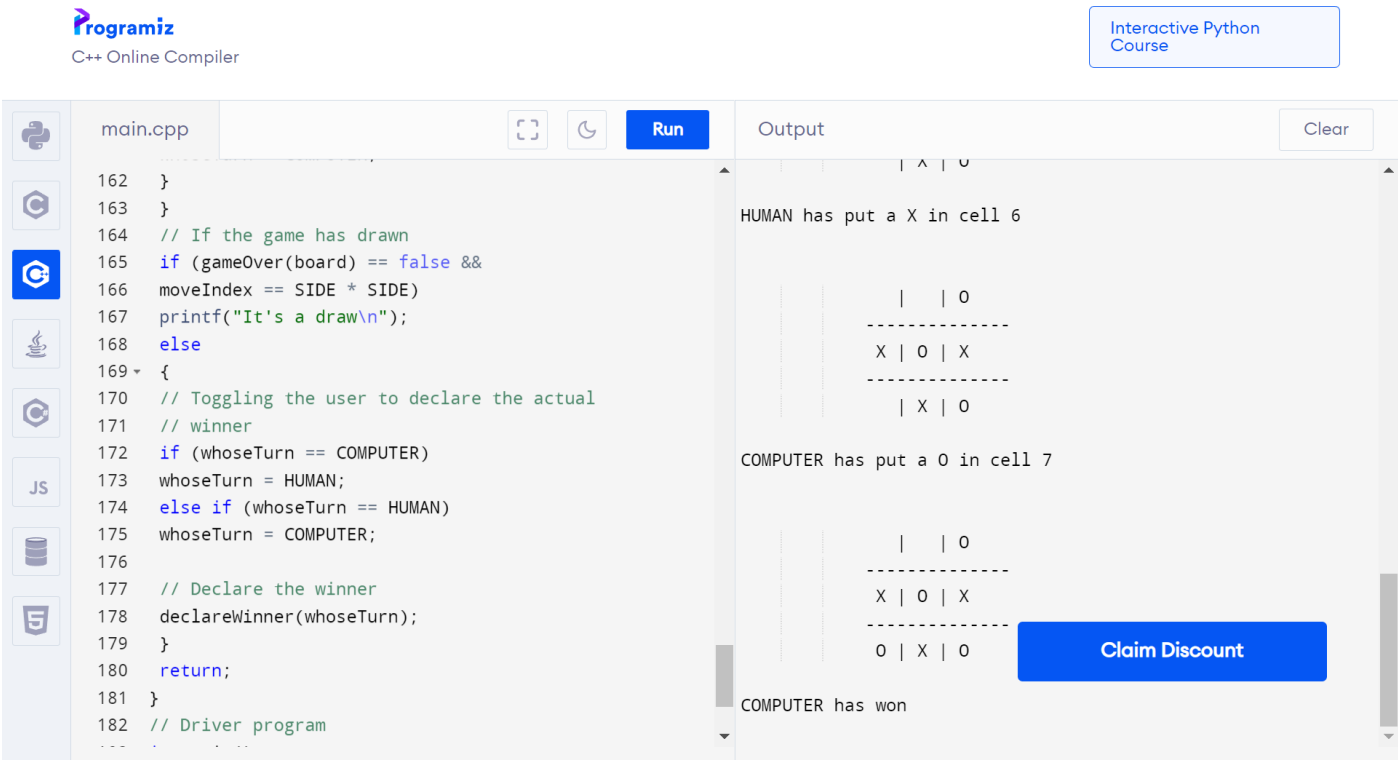
❖ Now computer has chosen the position 3rd position and human has chosen 4th position.



❖ Computer puts 0 in the cell 9th and now the human has chosen the position 6th in the table.



❖ At finally, the computer has taken the last chance and palced 0 in the position 7th And the game is drawn. Atlast the computer has won.



LEARNING OUTCOMES:

A lot of beginners and experienced programmers avoid learning Data

Structures and Algorithms because it's complicated and they think that there is no use of all the above stuff in real life but there is a lot of implementation of DSA in daily life. Data structures and algorithms (DSA) goes through solutions to standard problems in detail and gives you an insight into how efficient it is to use each one of them. It also teaches you the science of evaluating the efficiency of an algorithm. This enables you to choose the best of various choices.

For example If we have to search our roll number in 2000 pages of Document how would we do that? If we try to

search it randomly or in sequence it will take too much time. We can try another method in which we can directly go to page no. 1000 and we can see if our roll no. Is there or not if not we can move ahead and by repeating this and eliminating we can search our roll no. in no time.

And this is called Binary Search Algorithm.

Two reasons to Learn Data Structure and Algorithms -

If you want to crack the interviews and get into the product based companies If you love to solve the real-world complex problems. I have learnt a vast number of topics like Trees, Graphs, Linked Lists, Arrays, etc. I understood their basics, there working, there implementation, and their practical use in the problems we face while we solve a problem using coding. When we work in IT sector (Software or Programing part to be specific) we need to solve the problems and make programs write tons of code which will help us with the given problem and to write a program one need to make different algorithms. Many algorithms combine to make a program. Now, algorithm are writen in some lenguages but they are not dependen ton them, one need to make a plan and algo first then write it into any language wether i tis C++ or JAVA or C or any other programing language. Algorith is based on data structure and its implementation and working. So, basiclly one need to have a good grip on DSA to work in programing sector. A lot of newbie programmers have this question that where we use all the stuff of data structure and algorithm in our daily life and how it's useful in solving the real-world complex problem. We need to mention that whether you are interested in getting into the top tech giant companies or not DSA still helps a lot in your day-to-day life

What I Learned from the course precisely :

I Learned Data Structures and Algorithms from basic to advanced level.

Learned Topic-wise implementation of different Data Structures & Algorithms.

Improved my problem-solving skills to become a stronger developer.

Developed my analytical skills on Data Structures and use them efficiently.

Solved problems asked in product-based companies' interviews.

Solved problems in contests similar to coding round for SDE role.

CONCLUSION :

From learning dsa from geeks for geeks I have came to know that how much dsa play a role in a IT student life.By learning dsa we can solve all difficult problems by using various algorithms in real time applications also. ln playing Tic Tac Toe, there are some surprisingly strong life lessons to be learned. You can control other people. Not in a malicious way, but in a way that helps you guide your own choices and realize the impact your actions have on others. It has nothing to do with winning or losing.

"If you've chosen the right data structures and organized things well, the algorithms will almost always be self-evident. Data structures, not algorithms, are central to programming."

BIBLIOGRAPHY

DSA Books

Geeks for Geeks website

Geeks for Geeks Course

WEBLINKS:

https://practice.geeksforgeeks.org/batch/dsa-4