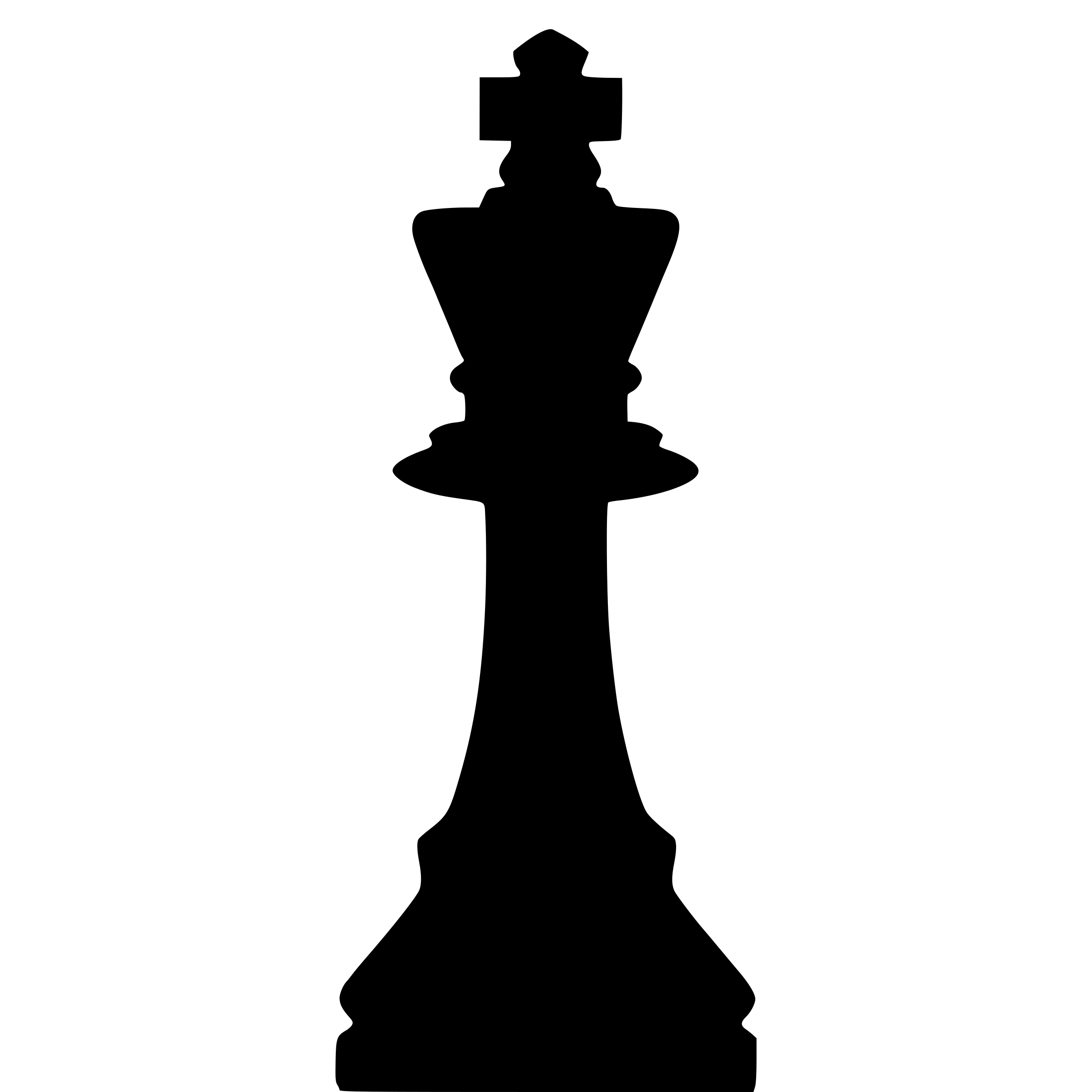
OBJECT ORIENTED PROGRAMMING

(Project)



QUEENS GAMBIT

SUBBMITTED TO: PROF.NOOR

SUBBMITED BY: GROUP NO.01

DEPARTEMENT: IT

SEMESTER: 02

COURSE CODE:IT-106

Project Proposal: QUEENS GAMBIT

***(FOR INVILGILATOR’S POURPOSE ONLY)***

**Objective:**

To create a basic of Chess Gambit in C++ following the complete principals of object oriented programming language that allows users to see what the pieces are in Chess and how to move the pieces on a digital board allowing the user to get himself familiar with the board rules and the math behind the movement of the pieces which can be further utilized to play special movements like FRENCH SUEDO, EN PASSANT and QUEENS GAMBIT.

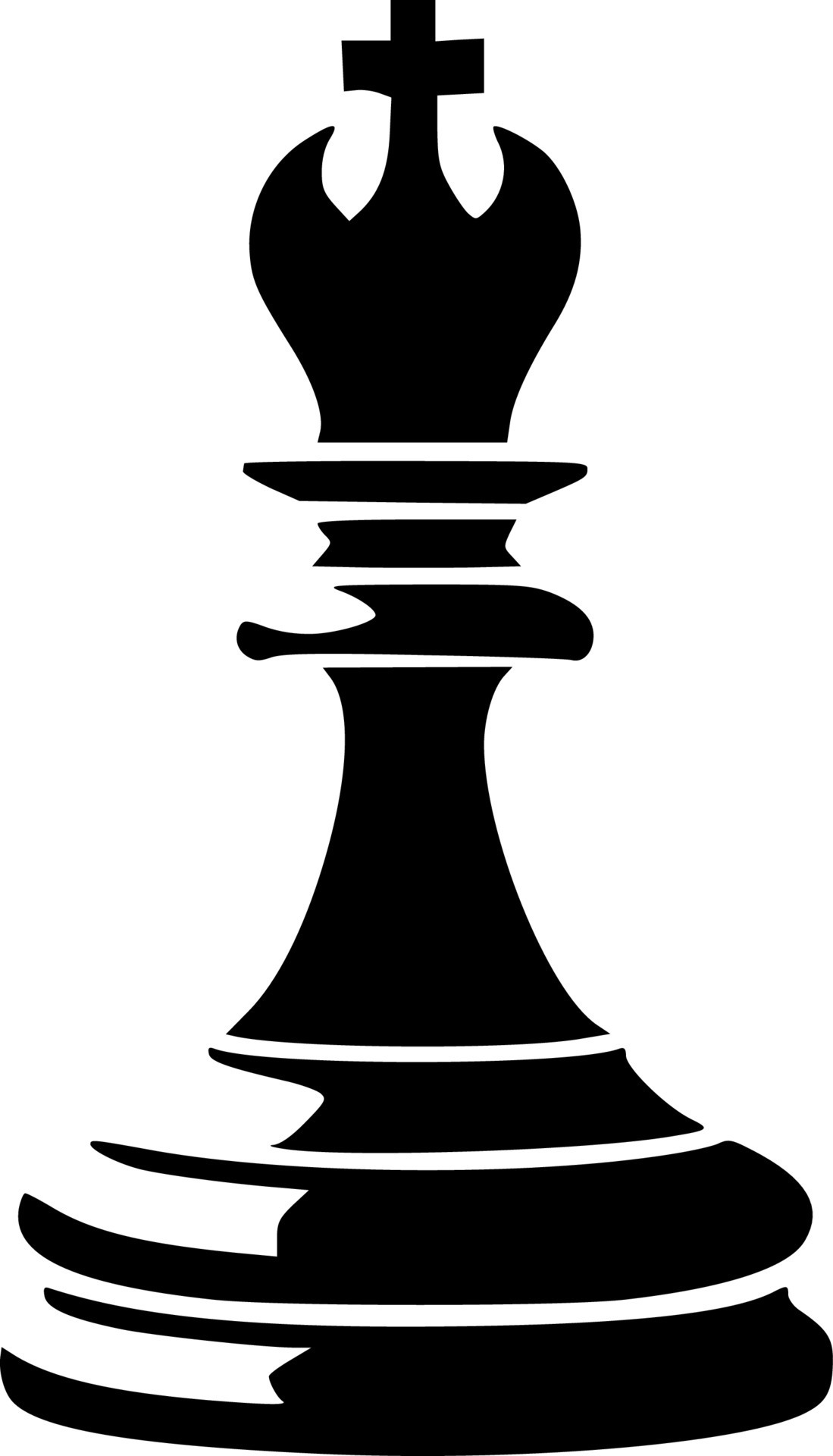
**Here are the basic objectives of the project:**

* Implement **core chess rules**, including piece movement and turns.
* Utilize **OOP principles** such as **inheritance, polymorphism, and encapsulation** to manage chess pieces.
* Develop **a functional board** that displays the game state in a console-based format.
* Enable **player interactions** by allowing moves to be input and processed dynamically.

**Team Distribution:**

**Member 1:**

Ideas gathering:

 Effectively found ideas regarding the project by physically buying a chess models and by visiting some of chess arenas to make the program more real problem solver piece of code. However, subject wasn’t well known about the chess scenarios but try hard to not only play a pivotal role in the project building but also to get himself loaded with chess logical rules.

Additionally, wrote pseudo-code for easily implementation of program to be possible also made the flow chart to control the logics within the program *(Written work would be provided on invigilator demands).*

**Member 2:**

Error findings and reviews interpreting:

By burning mid-nights oil this member effectively clear and solve all the logical errors programs was generating which leads to this error free code. Moreover, subject has played a vital role in this journey by connecting number of seniors and finding their reviews regarding the project not only this but also a proper and legitimate usage of AI to see beyond man’s capabilities.

**Member 3:**

Integration, Testing and Handling:

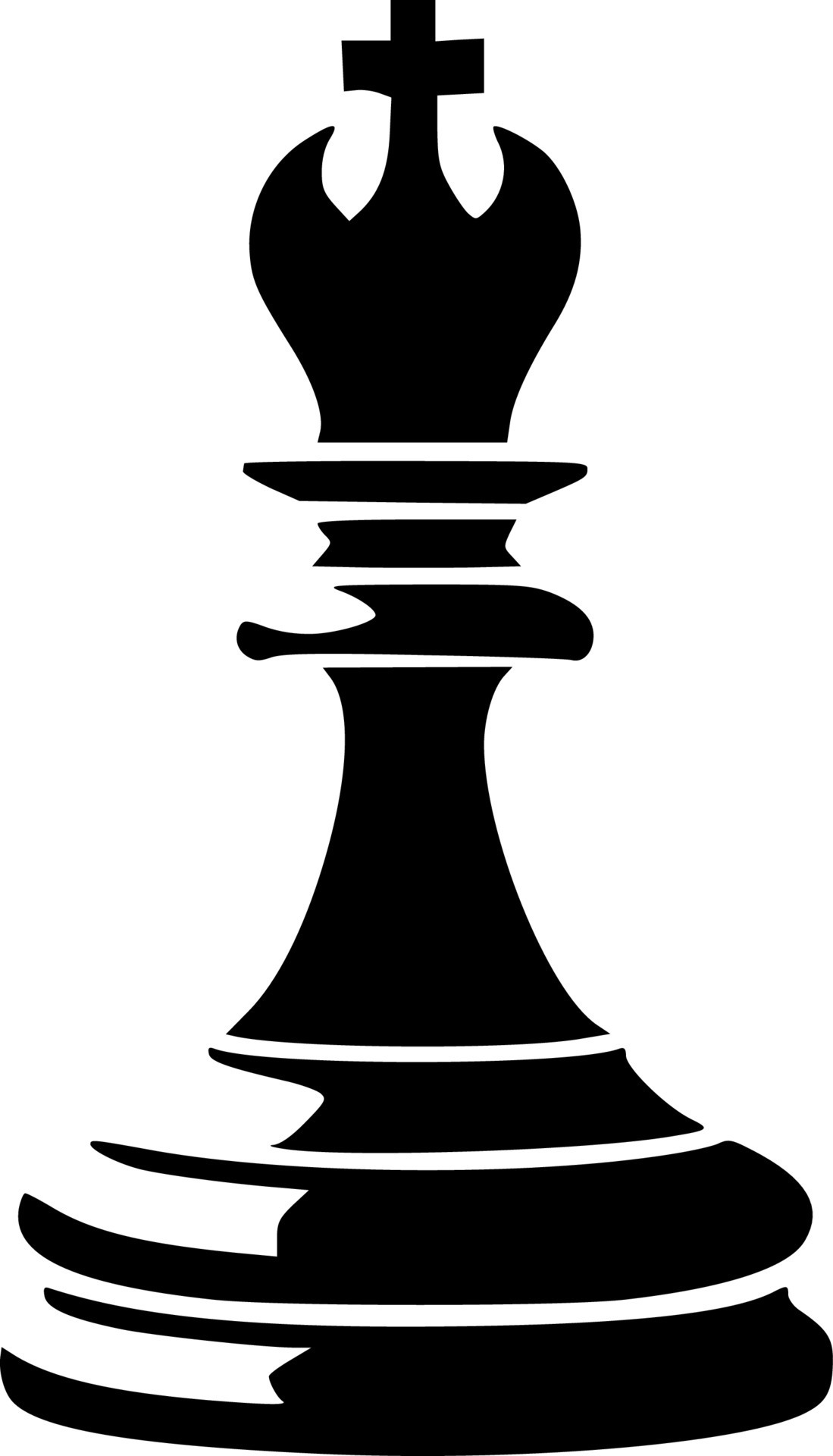
Supporting and guiding the team members like a light house top and helped the members to from going astray additionally ensuring the accuracy, reliability and effectiveness of the code by managing and distributing the work and chores equally on the bases of one’s capabilities.

As well as, developed a proper documentation that would allow and help the viewers to read and interpret the code and its outcomes without any hurdles.

***IN A NUTSHELL***

Each and every member helped and worked restlessly to make sure that their project will no less than best, each member regarding to his capabilities make sure their presence in all meetings and gatherings regarding to project and done their assigned work in specific given time.

***FUTURE CONSIDERATIONS***

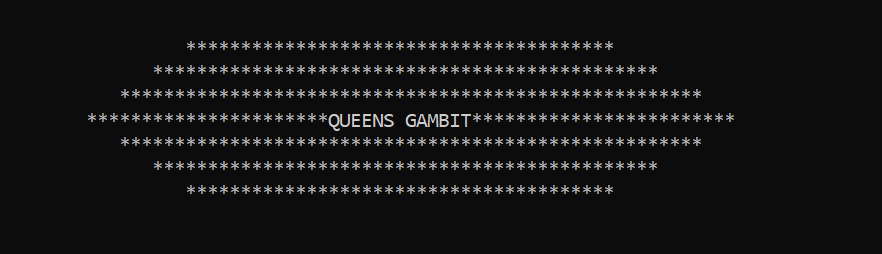
***Our goal is to developed a project that can carried to the next semesters and each semester adding up the functionalities and outcomes. Thus, the instead of multitasking we can work on a dedicated and specific piece of work***

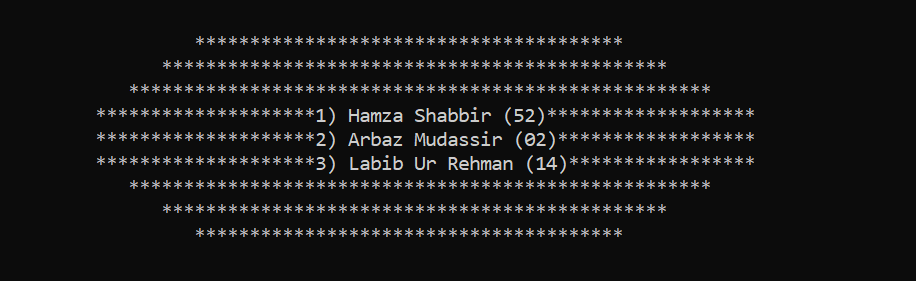
***Future Enhancements:***

Implement **check and checkmate detection**.

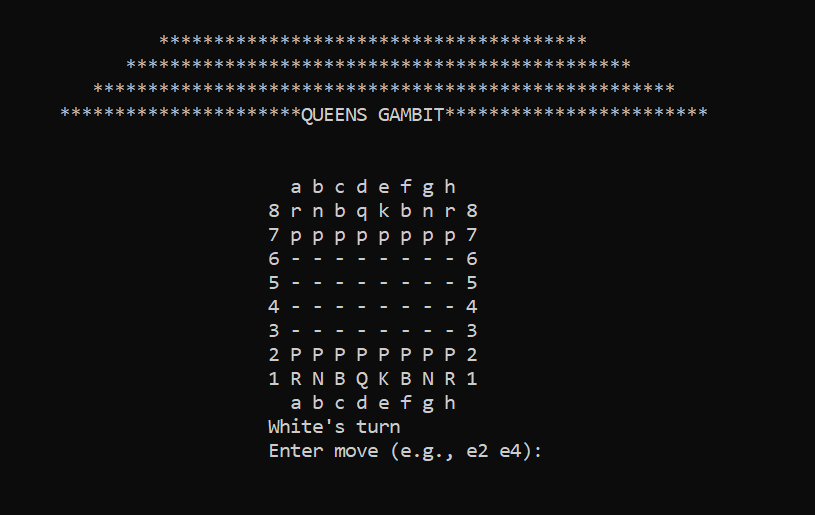
* Adding up the **stalemate conditions.**
* Implement specific **movement checks** and validation of moves.
* Add **graphical user interface (GUI)** for better visualization.
* Integrate **AI for single-player mode.**

FURTHER DETAILS





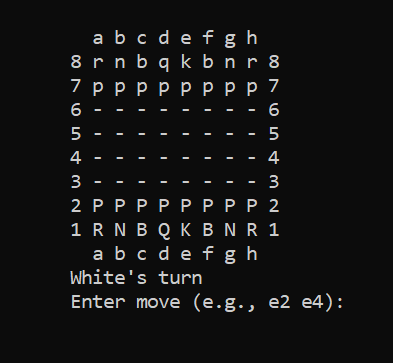
**View Panel**



*// This is what user will find as a main view on their screen which show’s a simulation of chess board [8\*8] having columns referred from alphabets (a-h) and rows from (1-8) which allows the user to move the pieces over the board.*

*//A two dimensional array is used to setup the board and a proper row and column guidance is visible so that user can directly enter the row and column correspondence integer values to move the pieces.*

**Understanding Pieces**



*//****Moreover, user would be able to see the pieces referred as:***

P for Pawn of color WHITE P **||** p for Pawn of color BLACK

R for Rook of color WHITE P **||** r for Rook of color BLACK

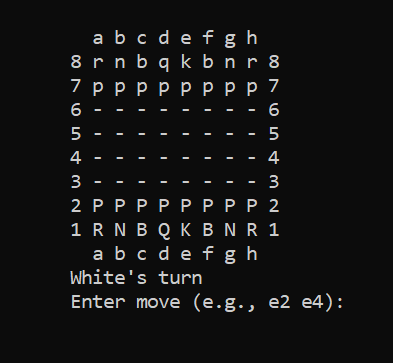
N for Knight of color WHITE P **||** n for Knight of color BLACK

B for Bishop of color WHITE P **||** b for Bishop of color BLACK

Q for Queen of color WHITE P **||** q for Queen of color BLACK

This Row (a-h) refers to Y axis.

(- - -) Shows empty squares over the board

 *Column of numbers (8-1) at X axis.*

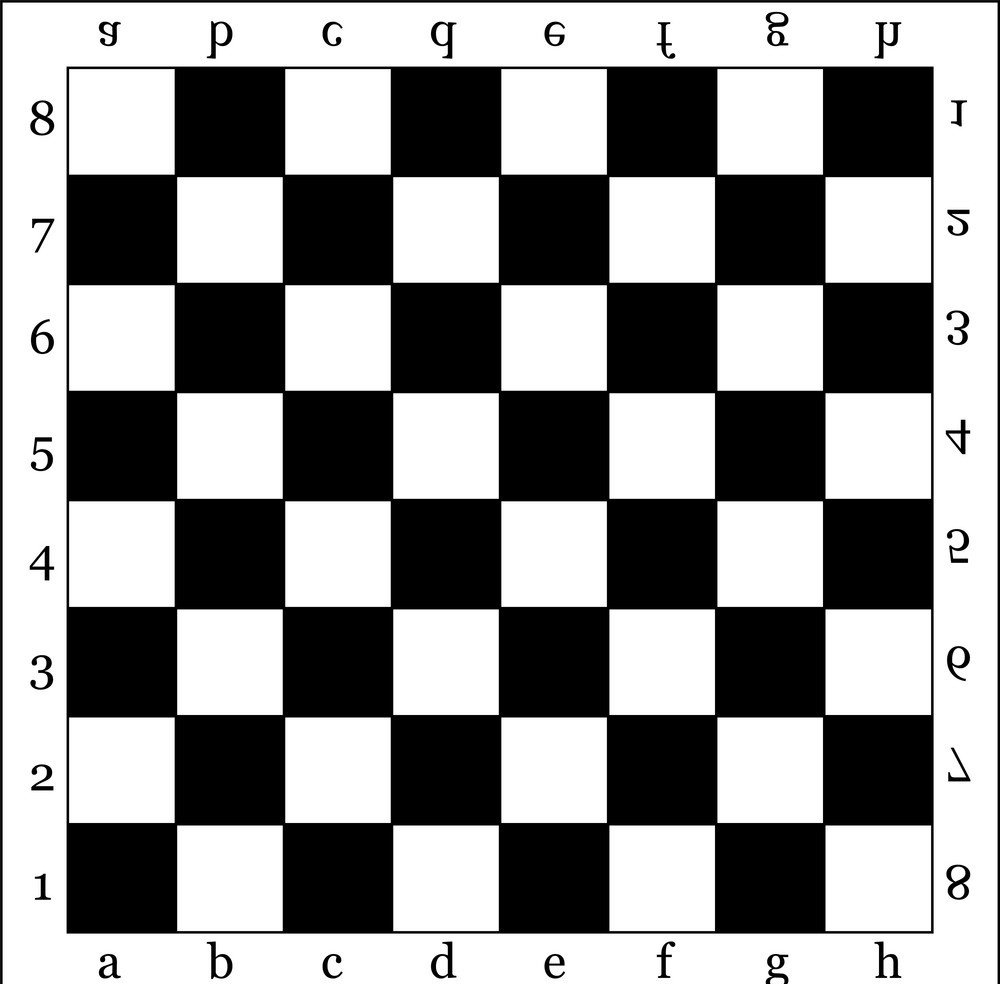
Specifies turns

*//Additionally, before prompting the user to enter the Move it is being specified by the code that whose turn is this as the game depends on two players therefore it is necessary to stay well known about the turns of Black or White teams.*

*//After specifying turn the code prompts the user to enter the move so the piece over the board is moved as the movement of the piece ultimately depends on user choice of commands.*

**Piece Movement**

***“How to move pieces”***



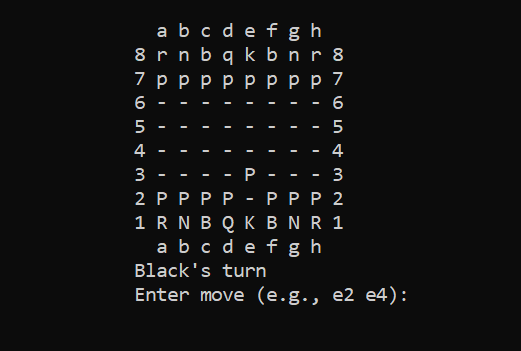
*//By examining the real chess b0ard picture, it is quite notable that the board is an inter-joining of rows and columns which are being specified as x axis and y axis*

*//Now if the user wants to move a piece only the cross ponding point of board which is supposed to new destination of that piece and the current location of that specific piece is to be given as an input for example:* ***If the user wants the piece Pawn referred as P to move forward on the board he only have to enter the current column and row number let say ‘e1’ and the destination location let say ‘e3’ and the piece would get relocated.***

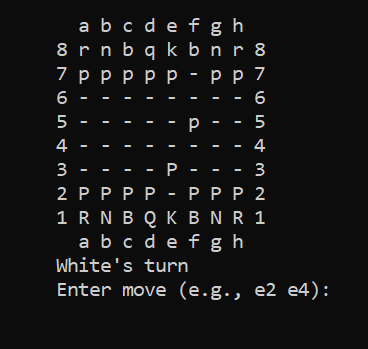
**Moving Pawn**

*//Pawn can move only one or two point (conditional) forward.*

***Input: (e2,e3)***



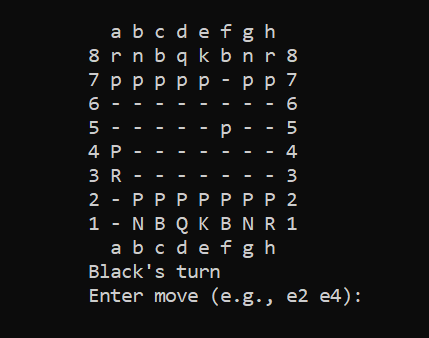
***Input: (f7,f5)***



**Moving Rook**

*//Moves straight in any direction*.

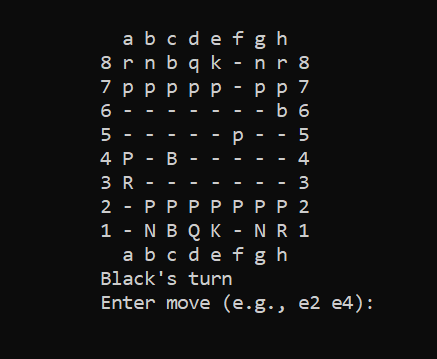
***Input: (a1,a3)***



**Moving Bishop**

***Input: (a1,a3)***

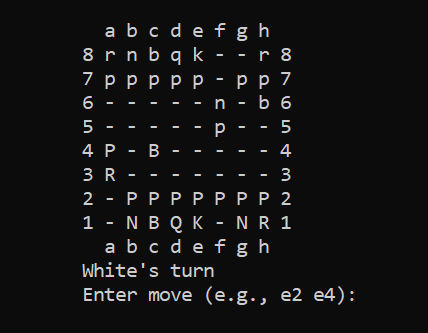
*//Bishop can diagonally move over the board.*



**Moving Knight**

***Input: (g8,f6)***

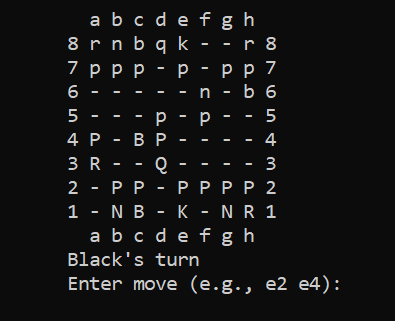
//*Moves in "L" shapes. It moves* ***two squares*** *in one direction and* ***one square*** *in the perpendicular direction.*



**Moving Queen**

***Input: (d1,d3)***

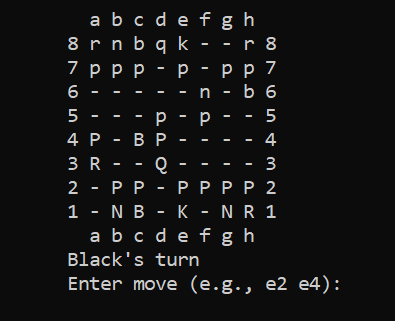
*//Moves like a rook or bishop*.



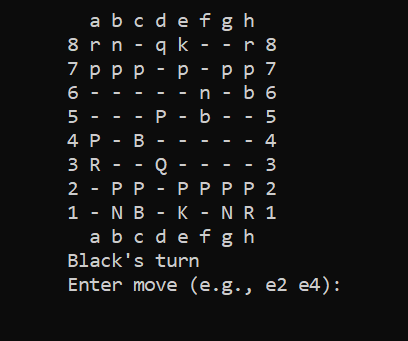
**Taking Pieces**

***“How to take pieces down”***

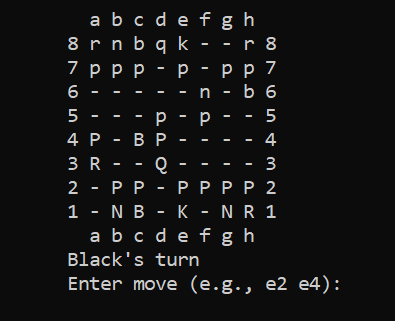
*//Let suppose you are playing as WHITE and got a hanging piece at location (d5) that should be taken down which can be done by moving your pawn standing at location (d4) to (d5)*



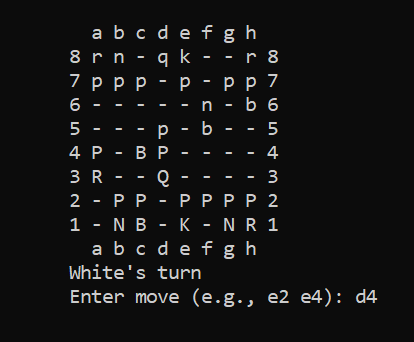
***//Piece (Black’s Pawn )taken down by White’s Pawn***



*//Let suppose you are playing as BLACK and got a hanging piece at location (F5) that should be taken down which cajn be done by moving your pawn standing at location (C8) to (d5)*



***//Piece (Whites’s Pawn )taken down by Black’s Bishop***



CODE C++

Code written in C++ language by following the concepts of Object Oriented Programming to handle all the working of this program which have plethora of benefits such as:

### **1. Encapsulation**

Encapsulation ensures that the internal workings of a class are **hidden** from outside access, exposing only **necessary functions**.

### **2. Inheritance**

Inheritance allows **child classes** to derive properties from a **parent class**.

### **3. Polymorphism**

Polymorphism allows methods to have **different implementations** in derived classes.

### **4. Abstraction**

Abstraction means **hiding** the **complexity** of the system while exposing only essential details.

### **5. Aggregation**

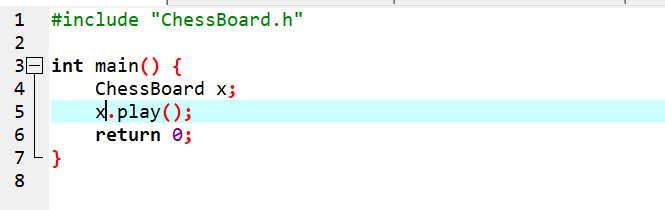
The **ChessBoard class** contains pieces, but pieces are **not inherently dependent** on the board for existence.

### **6. Dynamic Memory Management**

Out chess program **dynamically** allocates memory for pieces using new and properly delete them up using delete in the destructor ~ChessBoard(), preventing **memory leaks**.

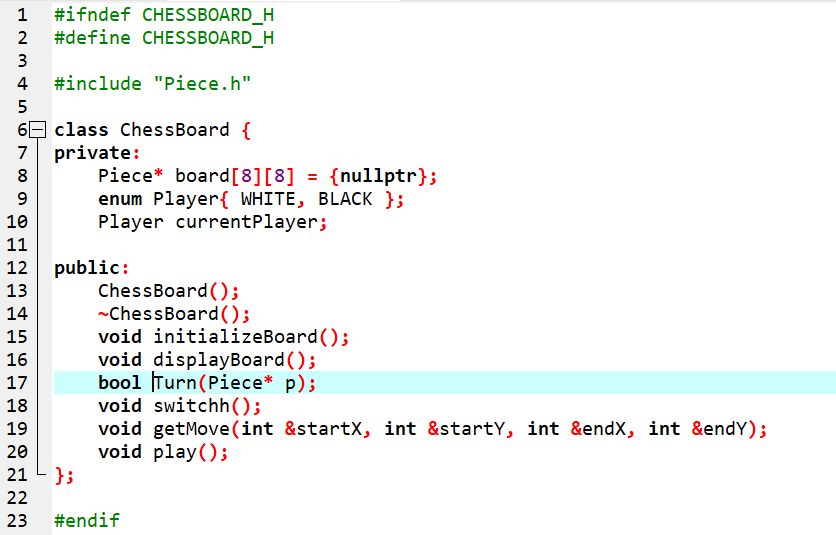
**Separations of Code:**

* **Main() Function** of the Code which basically runs the program or be stated as the entry point of the code



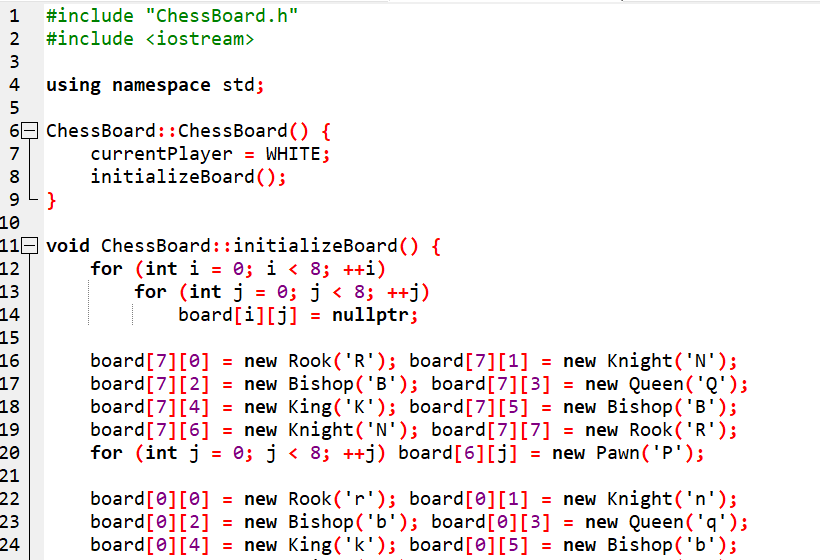
### **ChessBoard.h (Header File)**

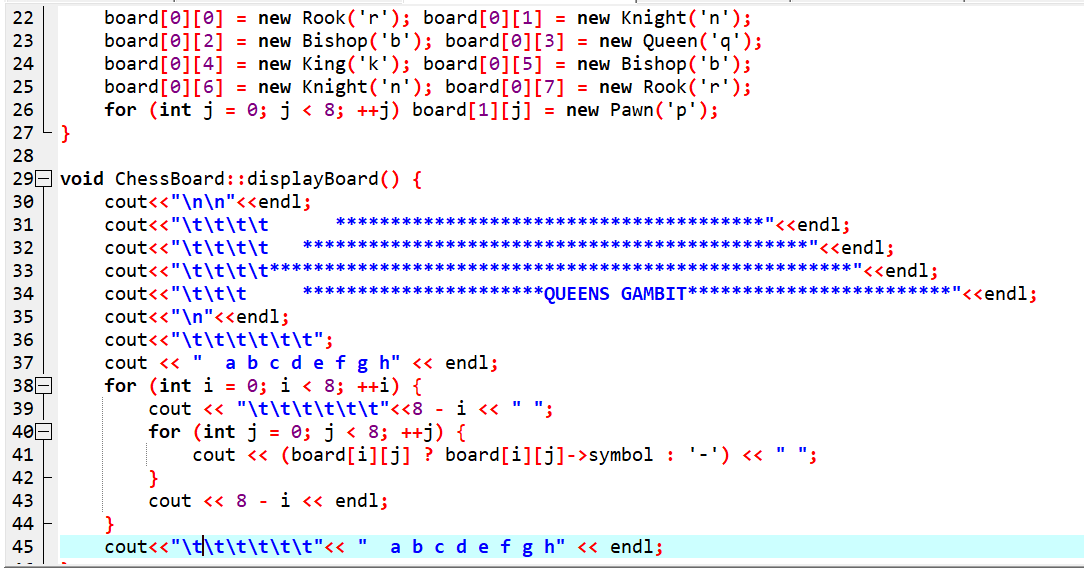
Declares the chessboard class.

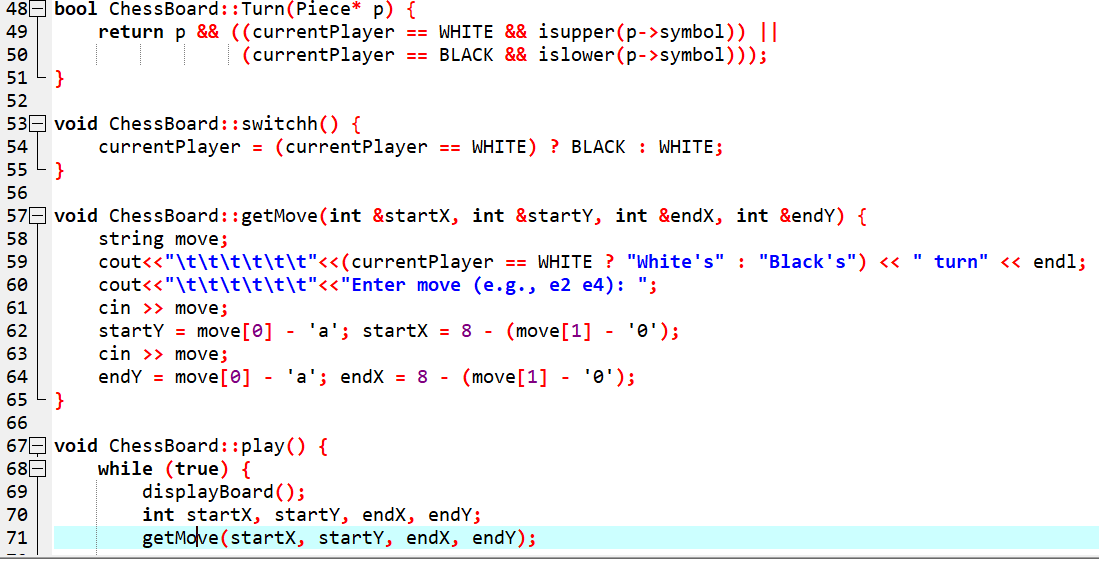


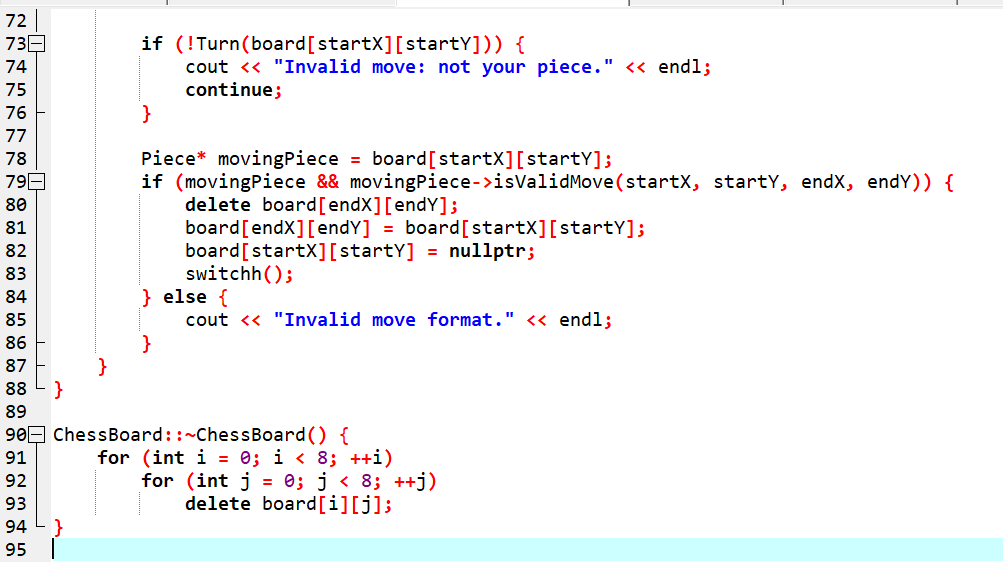
### **ChessBoard.cpp (Implementation File)**

Handles the game logic stated in the header file of chessboard.









### **Piece.h (Header File)**

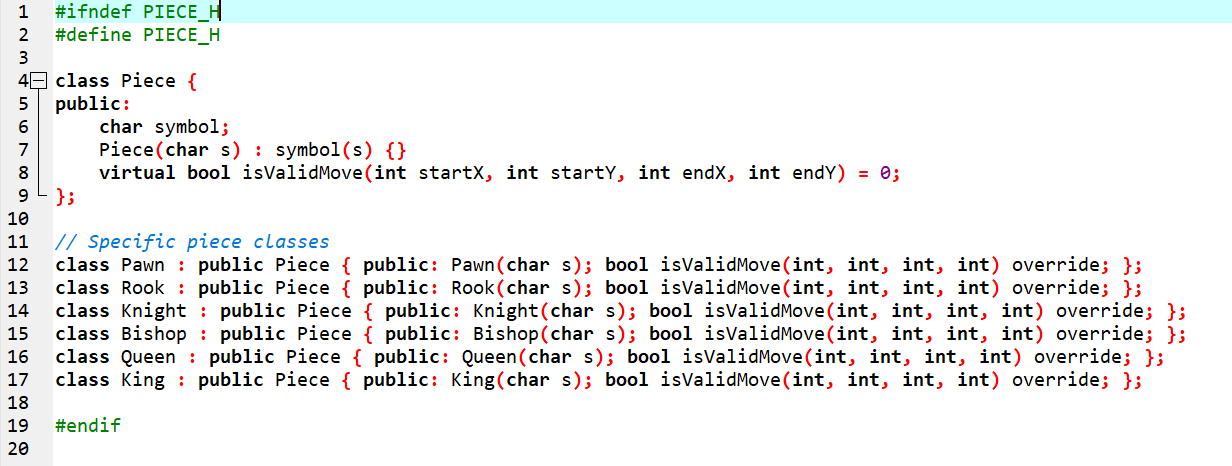
Defines the base class and specific piece types.

These classes are used to specify different types of pieces that are used to setup the chess board

An override function is used to follow the concepts of aggregation.

Bool data type is used to return 0/1 value as a counter check function.

These Daughter classes inherit from the parent class Piece

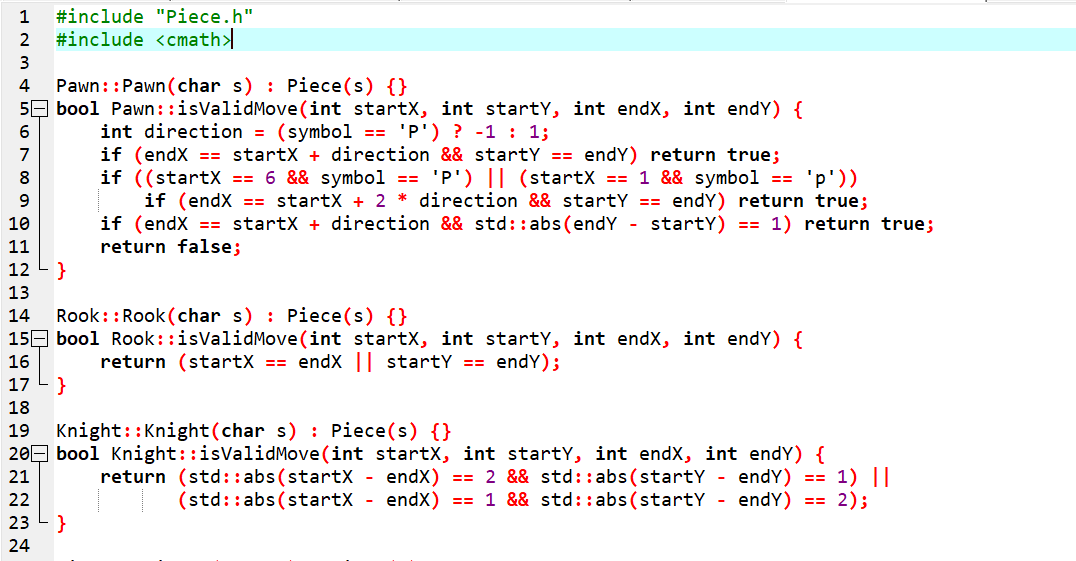


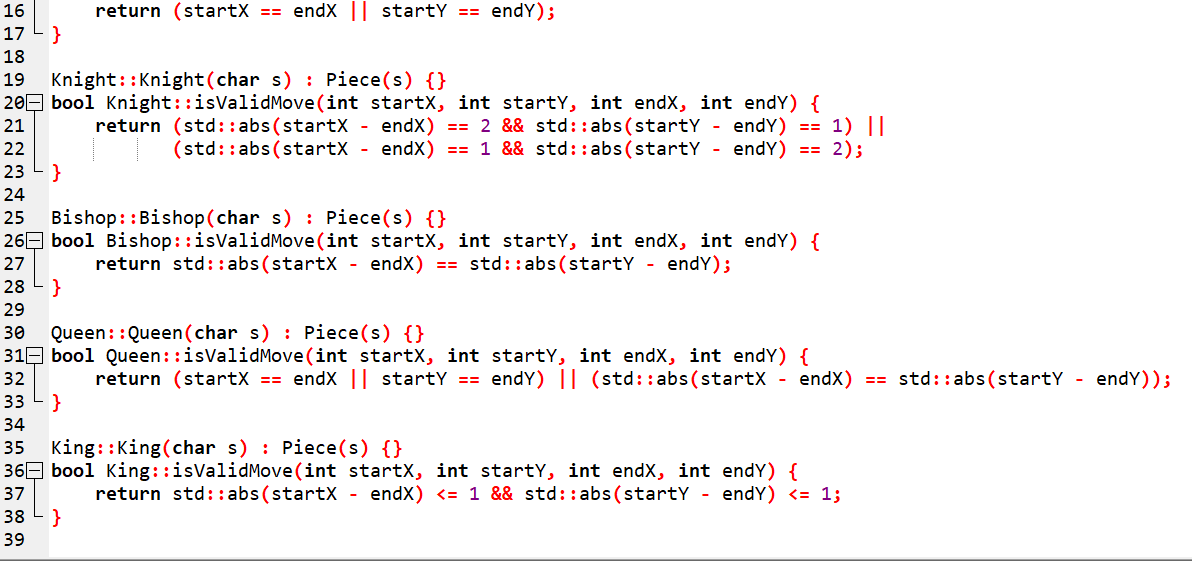
### **Piece.cpp (Implementation File)**

Defines how each piece moves implement constraints over the piece and limit their reach over the board.

Include cmath library for perfect working of the abs function use to calculate the absolute difference between two values asscii value.

Each of the piece have different condition as each movement is different from others.





//This code includes some limitation as this isn’t a complete simulation of chess gambit however this code can be furbished and shined to get an exact working model of chess