AI1

```
#include <iostream>
#include <list>
#include <map>
#include <queue>
using namespace std;
template <typename T>
class Graph {
  map<T, list<T>> adjList;
public:
  void addEdge(T src, T dest) {
    adjList[src].push_back(dest);
    adjList[dest].push_back(src);
  }
  void bfs(queue<T> &q, map<T, bool> &visited) {
    if (q.empty()) return;
    T node = q.front();
    q.pop();
    cout << node << " ";
    for (T n : adjList[node]) {
      if (!visited[n]) {
        q.push(n);
        visited[n] = true;
      }
    }
    bfs(q, visited);
  }
  void dfs(T v, map<T, bool> &visited) {
    visited[v] = true;
```

```
        cout << v << " ";

        for (T n : adjList[v]) {
            if (!visited[n]) {
                dfs(n, visited);
            }
        }
    }
};
int main() {
    Graph<int> g;
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(0, 3);
    g.addEdge(1, 4);
    g.addEdge(2, 5);
    g.addEdge(3, 6);
    g.addEdge(3, 7);
    queue<int> q;
    map<int, bool> visited;
    int ch;
    cout << "Enter 1 for BFS & 2 for DFS: ";
    cin >> ch;
    for (int i = 0; i < 8; i++) {
        visited[i] = false;
    }
    if (ch == 1) {
        // BFS traversal
        for (int i = 0; i < 8; i++) {
            if (!visited[i]) {
                visited[i] = true;
                q.push(i);
                g.bfs(q, visited);
            }
        }
        cout << endl;
    } else {
        for (int i = 0; i < 8; i++) {
            if (!visited[i]) {
                g.dfs(i, visited);
            }
        }
        cout << endl;
    }
    return 0;
}
```

AI2

```
import random
class TicTacToe:
    def __init__(self):
        self.board = []
    def create_board(self):
        for i in range(3):
            row = []
            for j in range(3):
                row.append('-')
            self.board.append(row)
```

```
def get_random_first_player(self):
    return random.randint(0, 1)
def fix_spot(self, row, col, player):
    self.board[row][col] = player
def is_player_win(self, player):
    n = len(self.board)
    for i in range(n):
        win = True
        for j in range(n):
            if self.board[i][j] != player:
                win = False
                break
        if win:
            return True
    for i in range(n):
        win = True
        for j in range(n):
            if self.board[j][i] != player:
                win = False
                break
        if win:
            return True
    win = True
    for i in range(n):
        if self.board[i][i] != player:
            win = False
            break
    if win:
        return True
    win = True
    for i in range(n):
        if self.board[i][n - 1 - i] != player:
            win = False
            break
    if win:
        return True


    return False
def is_board_filled(self):
    for row in self.board:
        for item in row:
            if item == '-':
                return False
    return True
def swap_player_turn(self, player):
    return 'X' if player == 'O' else 'O'
def show_board(self):
    for row in self.board:
        for item in row:
            print(item, end=" ")
        print()
def start(self):
    self.create_board()
    player = 'X' if self.get_random_first_player() == 1 else 'O'
    while True:
        print(f"Player {player} turn")
        self.show_board()
        row, col = list(map(int, input("Enter row and column numbers to fix spot: ").split()))
```

```
            print()
            self.fix_spot(row - 1, col - 1, player)
            if self.is_player_win(player):
                print(f"Player {player} wins the game!")
                break
            if self.is_board_filled():
                print("Match Draw!")
                break
            player = self.swap_player_turn(player)
        print()
        self.show_board()
tic_tac_toe = TicTacToe()
tic_tac_toe.start()
```

AI3

```
#include <iostream>
using namespace std;
int main() {
    int arr[10] = {6, 12, 0, 18, 11, 99, 55, 7, 44, 2};
    int n = 10;
    int i, j, pos, temp;
    for (i = 0; i < (n - 1); i++) {
        pos = i; // Set the initial position for the minimum value to i
        for (j = i + 1; j < n; j++) {
            if (arr[j] < arr[pos]) {
                pos = j;
            }
        }
        if (pos != i) {
            temp = arr[i];

            arr[i] = arr[pos];

            arr[pos] = temp;

        }

    }

    cout << "\n\t Sorted Array =\n\n\t";

    for (i = 0; i < n; i++) {

        cout << arr[i] << " ";

    }

    cout << endl;

    return 0;

}
```

AI4

```
#define N 4

#include <stdio.h>

/* Function to print solution */

void printSolution(int board[N][N]) {

    printf("\n\n Solution for 4 queen problem =\n\n");

    for (int i = 0; i < N; i++) {

        for (int j = 0; j < N; j++)

            printf(" %d ", board[i][j]);

        printf("\n");

    }

}

/* Function to check if a queen can be placed on board[row][col] */

int isSafe(int board[N][N], int row, int col) {

    int i, j;

    /* Check this row on left side */
    for (i = 0; i < col; i++)
        if (board[row][i])
            return 0;
    /* Check upper diagonal on left side */
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return 0;
    /* Check lower diagonal on left side */
    for (i = row, j = col; j >= 0 && i < N; i++, j--)
        if (board[i][j])
            return 0;
    return 1;
}
/* A recursive utility function to solve N Queen problem */
int solveNQUtil(int board[N][N], int col) {
    /* base case: If all queens are placed then return true */
    if (col >= N)
        return 1;
    /* Try placing this queen in all rows one by one */
    for (int i = 0; i < N; i++) {
        /* Check if the queen can be placed on board[i][col] */
        if (isSafe(board, i, col)) {
            /* Place this queen in board[i][col] */
            board[i][col] = 1;
            /* Recur to place rest of the queens */
            if (solveNQUtil(board, col + 1))
                return 1;
            /* If placing queen in board[i][col] doesn't lead to a solution
               then remove queen from board[i][col] */
            board[i][col] = 0; // BACKTRACK
        }
    }
    /* If the queen cannot be placed in any row in this column, return false */
    return 0;
}
/* This function solves the N Queen problem using Backtracking. */
int solveNQ() {
    int board[N][N] = { { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 },
                        { 0, 0, 0, 0 } };
    if (solveNQUtil(board, 0) == 0) {
        printf("Solution does not exist\n");
        return 0;
    }

    printSolution(board);
    return 1;
}
int main() {
    solveNQ();
    return 0;
}
```

AI5

```
def greet(bot_name, birth_year):
    print("Hello! My name is {0}.".format(bot_name))
    print("I was born in {0}.".format(birth_year))


def remind_name():
    print('Please, remind me your name.')
    name = input()
    print("What a great name you have, {0}!".format(name))


def guess_age():
    print('Let me guess your age.')
    print('Enter remainders of dividing your age by 3, 5 and 7.')
    rem3 = int(input())
    rem5 = int(input())
    rem7 = int(input())
    age = (rem3 * 70 + rem5 * 21 + rem7 * 15) % 105
    print("Your age is {0}; that's a good time to start programming!".format(age))


def count():
    print('Now I will prove to you that I can count to any number you want.')
    num = int(input())
    counter = 0
    while counter <= num:
        print("{0}!".format(counter))
        counter += 1


def test():
    print("Let's test your programming knowledge.")
    print("Why do we use methods?")
    print("1. To repeat a statement multiple times.")
    print("2. To decompose a program into several small subroutines.")
    print("3. To determine the execution time of a program.")
    print("4. To interrupt the execution of a program.")
    answer = 2
    guess = int(input())
    while guess != answer:
        print("Please, try again.")
        guess = int(input())
    print('Completed, have a nice day!')
    print('.................................')
    print('.................................')
    print('.................................')

def end():
    print('Congratulations, have a nice day!')
    print('.................................')
    print('.................................')
    print('.................................')
    input()

# Run the functions in order
greet('prachi', '2001')  # change it as needed
remind_name()
guess_age()
count()
test()
end()
```

AI6

```
#include <iostream>

#include <string>

using namespace std;

int main() {

   string patient_name;

   int patient_age;

   string patient_gender;

   bool has_fever;

   bool has_cough;

   bool has_sore_throat;

   bool has_difficulty_breathing;

   bool has_fatigue;

   bool has_headache;

   bool has_body_aches;

   bool has_diarrhea;

   bool has_loss_of_taste_or_smell;

   bool has_travel_history;

   bool has_contact_with_infected_person;

   string diagnosis;

   cout << "Welcome to the Hospital Diagnosis System. Please enter the following
information about the patient:" << endl;

   cout << "Name: ";

   getline(cin, patient_name);

   cout << "Age: ";

   cin >> patient_age;

   cout << "Gender (M/F): ";

   cin >> patient_gender;

   cout << "Does the patient have a fever? (1 for yes, 0 for no): ";
```

```
    cin >> has_fever;


    cout << "Does the patient have a cough? (1 for yes, 0 for no): ";
    cin >> has_cough;
    cout << "Does the patient have a sore throat? (1 for yes, 0 for no): ";
    cin >> has_sore_throat;
    cout << "Does the patient have difficulty breathing? (1 for yes, 0 for no): ";
    cin >> has_difficulty_breathing;
    cout << "Does the patient have fatigue? (1 for yes, 0 for no): ";
    cin >> has_fatigue;
    cout << "Does the patient have a headache? (1 for yes, 0 for no): ";
    cin >> has_headache;
    cout << "Does the patient have body aches? (1 for yes, 0 for no): ";
    cin >> has_body_aches;
    cout << "Does the patient have diarrhea? (1 for yes, 0 for no): ";
    cin >> has_diarrhea;
    cout << "Does the patient have a loss of taste or smell? (1 for yes, 0 for no): ";
    cin >> has_loss_of_taste_or_smell;
    cout << "Has the patient traveled recently? (1 for yes, 0 for no): ";
    cin >> has_travel_history;
    cout << "Has the patient come in contact with someone infected with COVID-19? (1 for
yes, 0 for no): ";
    cin >> has_contact_with_infected_person;
    // Check for common symptoms and determine the diagnosis
    if (has_difficulty_breathing) {
        diagnosis = "Severe Acute Respiratory Syndrome (SARS)";
    } else if (has_fever && (has_cough || has_sore_throat || has_difficulty_breathing)) {
        diagnosis = "Coronavirus Disease 2019 (COVID-19)";
    } else if (has_fever && (has_headache || has_body_aches || has_fatigue)) {
        diagnosis = "Influenza (Flu)";
    } else {
        diagnosis = "Common Cold";
    }
    // Output the diagnosis to the user
    cout << endl;
    cout << "Diagnosis for " << patient_name << ":" << endl;
    cout << "Age: " << patient_age << endl;
    cout << "Gender: " << patient_gender << endl;
    cout << "Diagnosis: " << diagnosis << endl;
    return 0;
}
```