MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE GWALIOR
(A Govt. Aided UGC Autonomous Institute Affiliated to RGPV, Bhopal)
NAAC Accredited with A++ Grade

Skill Based Mini Project Report

on

PROBLEM SOLVING AND PROGRAMMING (3280122)

SUBMITTED BY

Shalini Varfa & Yuvraj Shukla
0901AM231059 & 0901AM231077
I semester
Artificial Intelligence and Machine Learning

SUBMITTED TO

Dr. Mir Shahnawaz Ahmad
Assistant Professor

CENTRE FOR ARTIFICIAL INTELLIGENCE
Madhav Institute of Technology & Science
Gwalior - 474005 (MP) est. 1957

Session: 2023-24

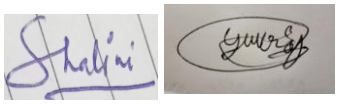
DECLARATION

I hereby declare that the mini skill-based project for the course **Problem Solving and Programming (3280122)** is being submitted in the partial fulfilment of the requirement for the award of **Bachelor of Technology** in **Artificial Intelligence and Machine Learning.**

All the information in this document has been obtained and presented in accordance with academic rule and ethical conduct.

Date :

Place: Gwalior



Shalini Varfa&
Yuvraj Shukla
0901AM23017709
01AM2301770901
AM230177

1} **Create a simple banking system that allows users to open accounts, deposit and withdraw money,**
and check their account balances.

```
#include <iostream>
#include <iomanip>
#include <map>

using namespace std;

class Bank {
private:
   map<int, double> accounts;
   int accountNumberCounter;

public:
   Bank() : accountNumberCounter(1000) {}

   // Open a new account
   void openAccount() {
      accounts[accountNumberCounter] = 0.0;
      cout << "Account created successfully. Your account number is: " << accountNumberCounter <<
endl;
      accountNumberCounter++;
   }

   // Deposit money into an account
   void deposit(int accountNumber, double amount) {
      if (accounts.find(accountNumber) != accounts.end()) {
         accounts[accountNumber] += amount;
         cout << "Deposit successful. New balance: $" << fixed << setprecision(2) <<
accounts[accountNumber] << endl;
      } else {
         cout << "Account not found." << endl;
      }
   }

   // Withdraw money from an account
   void withdraw(int accountNumber, double amount) {
      if (accounts.find(accountNumber) != accounts.end()) {
         if (accounts[accountNumber] >= amount) {
            accounts[accountNumber] -= amount;
            cout << "Withdrawal successful. New balance: $" << fixed << setprecision(2) <<
accounts[accountNumber] << endl;
         } else {
            cout << "Insufficient funds." << endl;
         }
      } else {
         cout << "Account not found." << endl;
```

```
        }
    }

    // Check account balance
    void checkBalance(int accountNumber) {
        if (accounts.find(accountNumber) != accounts.end()) {
            cout << "Account balance: $" << fixed << setprecision(2) << accounts[accountNumber] <<
endl;
        } else {
            cout << "Account not found." << endl;
        }
    }
};

int main() {
    Bank bank;
    int choice;
    int accountNumber;
    double amount;

    do {
        cout << "\nBanking System Menu:\n";
        cout << "1. Open Account\n";
        cout << "2. Deposit\n";
        cout << "3. Withdraw\n";
        cout << "4. Check Balance\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                bank.openAccount();
                break;
            case 2:
                cout << "Enter account number: ";
                cin >> accountNumber;
                cout << "Enter amount to deposit: $";
                cin >> amount;
                bank.deposit(accountNumber, amount);
                break;
            case 3:
                cout << "Enter account number: ";
                cin >> accountNumber;
                cout << "Enter amount to withdraw: $";
                cin >> amount;
                bank.withdraw(accountNumber, amount);
                break;
            case 4:
                cout << "Enter account number: ";
                cin >> accountNumber;
```

```
            bank.checkBalance(accountNumber);
            break;
        case 5:
            cout << "Exiting the program. Thank you!\n";
            break;
        default:
            cout << "Invalid choice. Please try again.\n";
    }

  } while (choice != 5);

  return 0;
}
```

Build a two-player console-based Tic-Tac-Toe game where players take turns marking a 3x3 grid.

```
#include <iostream>

using namespace std;

// Function to print the Tic-Tac-Toe board
void printBoard(char board[3][3]) {
   cout << "  1 2 3\n";
   for (int i = 0; i < 3; ++i) {
      cout << i + 1 << " ";
      for (int j = 0; j < 3; ++j) {
         cout << board[i][j] << " ";
      }
      cout << endl;
   }
   cout << endl;
}

// Function to check if a player has won
bool checkWin(char board[3][3], char player) {
   // Check rows and columns
   for (int i = 0; i < 3; ++i) {
      if ((board[i][0] == player && board[i][1] == player && board[i][2] == player) ||
         (board[0][i] == player && board[1][i] == player && board[2][i] == player)) {
         return true;
      }
   }

   // Check diagonals
   if ((board[0][0] == player && board[1][1] == player && board[2][2] == player) ||
      (board[0][2] == player && board[1][1] == player && board[2][0] == player)) {
      return true;
   }

   return false;
}

// Function to check if the board is full (a tie)
bool checkTie(char board[3][3]) {
   for (int i = 0; i < 3; ++i) {
      for (int j = 0; j < 3; ++j) {
         if (board[i][j] == ' ') {
            return false; // Empty space found, game is not a tie
         }
      }
   }
   return true; // Board is full, game is a tie
```

```
}

int main() {
    char board[3][3] = {{' ', ' ', ' '}, {' ', ' ', ' '}, {' ', ' ', ' '}};
    char currentPlayer = 'X';

    while (true) {
        printBoard(board);

        // Get player move
        int row, col;
        cout << "Player " << currentPlayer << ", enter your move (row and column): ";
        cin >> row >> col;

        // Validate the move
        if (row < 1 || row > 3 || col < 1 || col > 3 || board[row - 1][col - 1] != ' ') {
            cout << "Invalid move. Try again.\n";
            continue;
        }

        // Make the move
        board[row - 1][col - 1] = currentPlayer;

        // Check for a win
        if (checkWin(board, currentPlayer)) {
            printBoard(board);
            cout << "Player " << currentPlayer << " wins!\n";
            break;
        }

        // Check for a tie
        if (checkTie(board)) {
            printBoard(board);
            cout << "It's a tie!\n";
            break;
        }

        // Switch to the other player
        currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
    }

    return 0;
}
```

Develop an address book application to store and manage contacts' names, phone numbers, and addresses.

```
#include <iostream>
#include <string>

using namespace std;

const int MAX_CONTACTS = 100;

struct Contact {
   string name;
   string phoneNumber;
   string address;
};

class AddressBook {
private:
   Contact contacts[MAX_CONTACTS];
   int contactCount;

public:
   AddressBook() : contactCount(0) {}

   void addContact(const Contact& contact) {
      if (contactCount < MAX_CONTACTS) {
         contacts[contactCount++] = contact;
         cout << "Contact added successfully." << endl;
      } else {
         cout << "Address book is full. Cannot add more contacts." << endl;
      }
   }

   void displayContacts() const {
      if (contactCount == 0) {
         cout << "Address book is empty." << endl;
         return;
      }

      cout << "Contacts:" << endl;
      for (int i = 0; i < contactCount; ++i) {
         cout << "Name: " << contacts[i].name << endl;
         cout << "Phone: " << contacts[i].phoneNumber << endl;
         cout << "Address: " << contacts[i].address << endl << endl;
      }
   }

   void searchContact(const string& searchTerm) const {
      bool found = false;
```

```
        for (int i = 0; i < contactCount; ++i) {
            if (contacts[i].name.find(searchTerm) != string::npos ||
                contacts[i].phoneNumber.find(searchTerm) != string::npos ||
                contacts[i].address.find(searchTerm) != string::npos) {
                cout << "Name: " << contacts[i].name << endl;
                cout << "Phone: " << contacts[i].phoneNumber << endl;
                cout << "Address: " << contacts[i].address << endl << endl;
                found = true;
            }
        }

        if (!found) {
            cout << "Contact not found." << endl;
        }
    }
};

int main() {
    AddressBook addressBook;

    while (true) {
        cout << "1. Add Contact" << endl;
        cout << "2. Display Contacts" << endl;
        cout << "3. Search Contact" << endl;
        cout << "4. Exit" << endl;
        cout << "Enter your choice: ";

        int choice;
        cin >> choice;

        switch (choice) {
            case 1: {
                Contact newContact;
                cin.ignore();

                cout << "Enter Name: ";
                getline(cin, newContact.name);

                cout << "Enter Phone Number: ";
                getline(cin, newContact.phoneNumber);

                cout << "Enter Address: ";
                getline(cin, newContact.address);

                addressBook.addContact(newContact);
                break;
            }
            case 2:
                addressBook.displayContacts();
                break;
            case 3: {
```

```
            cin.ignore();

            cout << "Enter search term: ";
            string searchTerm;
            getline(cin, searchTerm);

            addressBook.searchContact(searchTerm);
            break;
        }
        case 4:
            cout << "Exiting the address book application. Goodbye!" << endl;
            return 0;
        default:
            cout << "Invalid choice. Please enter a valid option." << endl;
      }
    }

    return 0;
}
```

Create a program to manage student records, including information like name, ID, and grades. Users can add, update, and view student data.

```
#include <iostream>
#include <string>

using namespace std;

const int MAX_STUDENTS = 100;

struct Student {
   string name;
   int id;
   float grades;
};

class StudentManager {
private:
   Student students[MAX_STUDENTS];
   int studentCount;

public:
   StudentManager() : studentCount(0) {}

   void addStudent(const Student& student) {
      if (studentCount < MAX_STUDENTS) {
         students[studentCount++] = student;
         cout << "Student added successfully." << endl;
      } else {
         cout << "Maximum number of students reached. Cannot add more students." << endl;
      }
   }

   void updateStudent(int studentId, const Student& updatedStudent) {
      for (int i = 0; i < studentCount; ++i) {
         if (students[i].id == studentId) {
            students[i] = updatedStudent;
            cout << "Student information updated successfully." << endl;
            return;
         }
      }

      cout << "Student with ID " << studentId << " not found." << endl;
   }

   void displayStudents() const {
```

```
        if (studentCount == 0) {
            cout << "No student records available." << endl;
            return;
        }

        cout << "Student Records:" << endl;
        for (int i = 0; i < studentCount; ++i) {
            cout << "ID: " << students[i].id << endl;
            cout << "Name: " << students[i].name << endl;
            cout << "Grades: " << students[i].grades << endl << endl;
        }
    }
};

int main() {
    StudentManager studentManager;

    while (true) {
        cout << "1. Add Student" << endl;
        cout << "2. Update Student" << endl;
        cout << "3. View Students" << endl;
        cout << "4. Exit" << endl;
        cout << "Enter your choice: ";

        int choice;
        cin >> choice;

        switch (choice) {
            case 1: {
                Student newStudent;
                cin.ignore(); // Clear the newline character from the buffer

                cout << "Enter Name: ";
                getline(cin, newStudent.name);

                cout << "Enter ID: ";
                cin >> newStudent.id;

                cout << "Enter Grades: ";
                cin >> newStudent.grades;

                studentManager.addStudent(newStudent);
                break;
            }
            case 2: {
                int studentId;
                cout << "Enter the ID of the student to update: ";
                cin >> studentId;

                Student updatedStudent;
                cin.ignore();
```

```
            cout << "Enter Updated Name: ";
            getline(cin, updatedStudent.name);

            cout << "Enter Updated Grades: ";
            cin >> updatedStudent.grades;

            studentManager.updateStudent(studentId, updatedStudent);
            break;
        }
        case 3:
            studentManager.displayStudents();
            break;
        case 4:
            cout << "Exiting the student records management program. Goodbye!" << endl;
            return 0;
        default:
            cout << "Invalid choice. Please enter a valid option." << endl;
    }
  }

  return 0;
}
```