**CODE ALPHA**

**Task no 1:**

#include <iostream>

#include <vector>

using namespace std;

int main() {

int n;

cout << "Enter number of courses: ";

cin >> n;

vector<double> credits(n), grades(n);

double totalCredits = 0, totalGradePoints = 0;

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

cout << "Enter credits for course " << i + 1 << ": ";

cin >> credits[i];

cout << "Enter grade point (0-10) for course " << i + 1 << ": ";

cin >> grades[i];

totalCredits += credits[i];

totalGradePoints += credits[i] \* grades[i];

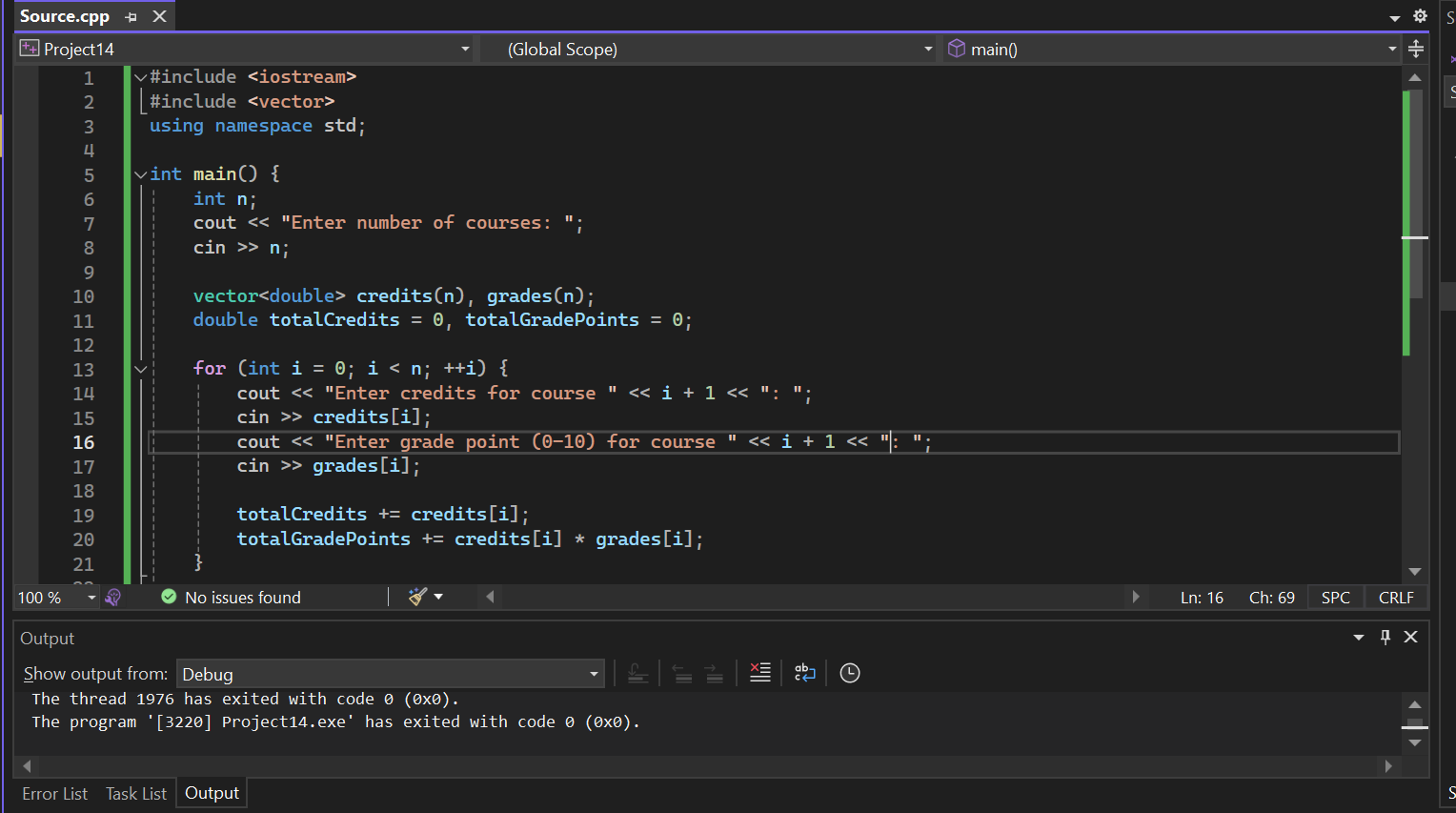
}

double cgpa = totalGradePoints / totalCredits;

cout << "Your CGPA is: " << cgpa << endl;

return 0;

}



Task no 2:

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

void registerUser() {

string username, password;

cout << "Enter username: ";

cin >> username;

cout << "Enter password: ";

cin >> password;

ofstream file(username + ".txt");

file << username << endl << password;

file.close();

cout << "Registration successful!\n";

}

void loginUser() {

string username, password, user, pass;

cout << "Enter username: ";

cin >> username;

cout << "Enter password: ";

cin >> password;

ifstream file(username + ".txt");

if (file.is\_open()) {

getline(file, user);

getline(file, pass);

if (user == username && pass == password)

cout << "Login successful!\n";

else

cout << "Invalid credentials.\n";

} else {

cout << "User not found.\n";

}

}

int main() {

int choice;

cout << "1. Register\n2. Login\nEnter choice: ";

cin >> choice;

if (choice == 1)

registerUser();

else if (choice == 2)

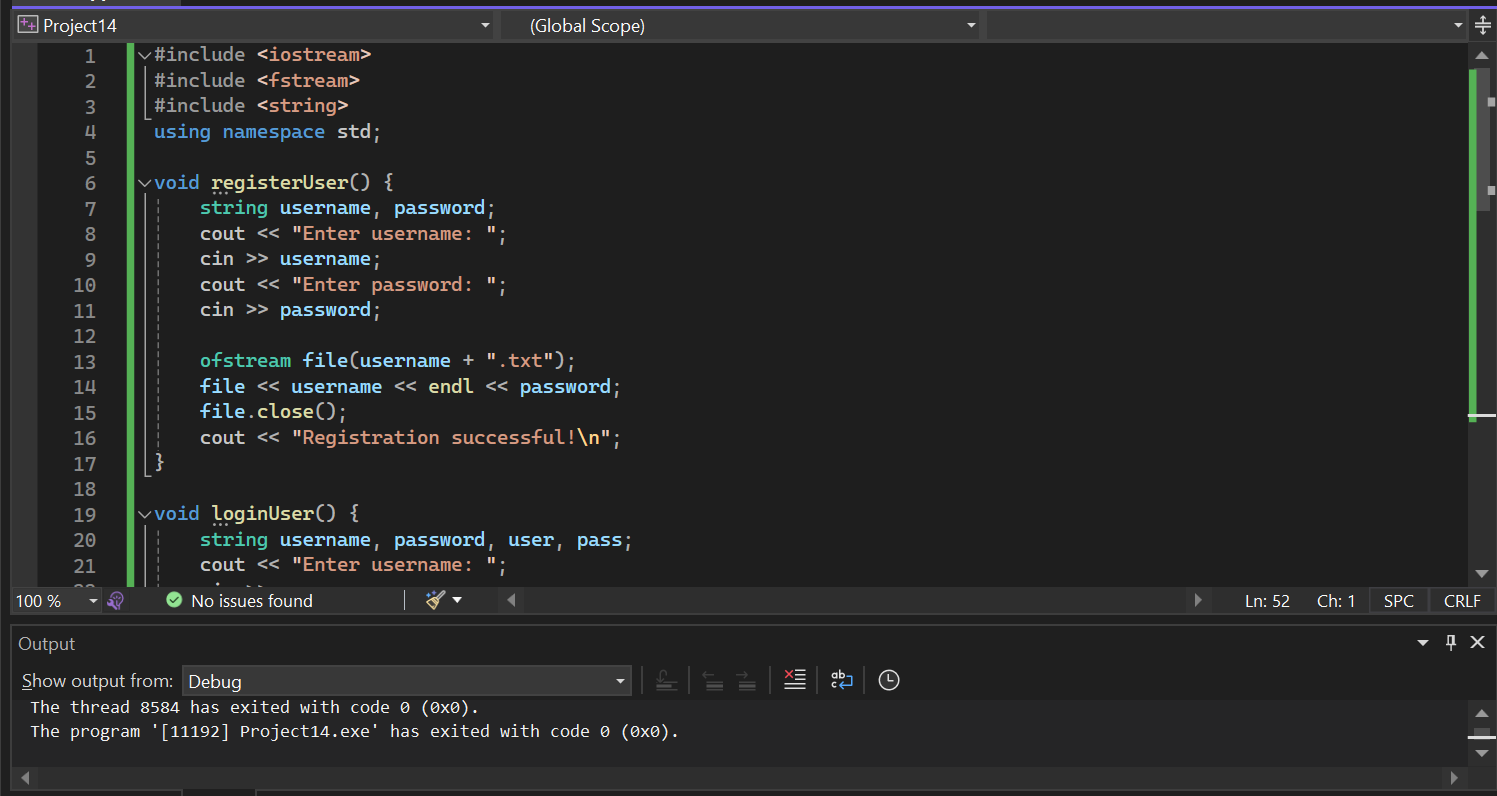
loginUser();

else

cout << "Invalid choice.\n";

return 0;

}



Task 4:

#include <iostream>

#include <map>

using namespace std;

class Account {

string name;

int accNo;

double balance;

public:

Account() : name(""), accNo(0), balance(0) {} // Added default constructor

Account(string n, int a, double b) : name(n), accNo(a), balance(b) {}

void deposit(double amt) {

balance += amt;

cout << "Deposited: " << amt << endl;

}

void withdraw(double amt) {

if (amt > balance)

cout << "Insufficient balance!\n";

else {

balance -= amt;

cout << "Withdrawn: " << amt << endl;

}

}

void show() {

cout << "Account No: " << accNo << ", Name: " << name << ", Balance: " << balance << endl;

}

};

map<int, Account> database;

int main() {

int choice, accNo;

string name;

double amt;

while (true) {

cout << "\n1. Create Account\n2. Deposit\n3. Withdraw\n4. Show Account\n5. Exit\nChoice: ";

cin >> choice;

if (choice == 1) {

cout << "Enter Account No: ";

cin >> accNo;

cout << "Enter Name: ";

cin >> name;

cout << "Enter Initial Deposit: ";

cin >> amt;

database[accNo] = Account(name, accNo, amt);

}

else if (choice == 2) {

cout << "Enter Account No: ";

cin >> accNo;

if (database.find(accNo) == database.end()) {

cout << "Account not found!\n";

continue;

}

cout << "Enter amount to deposit: ";

cin >> amt;

database[accNo].deposit(amt);

}

else if (choice == 3) {

cout << "Enter Account No: ";

cin >> accNo;

if (database.find(accNo) == database.end()) {

cout << "Account not found!\n";

continue;

}

cout << "Enter amount to withdraw: ";

cin >> amt;

database[accNo].withdraw(amt);

}

else if (choice == 4) {

cout << "Enter Account No: ";

cin >> accNo;

if (database.find(accNo) == database.end()) {

cout << "Account not found!\n";

continue;

}

database[accNo].show();

}

else if (choice == 5)

break;

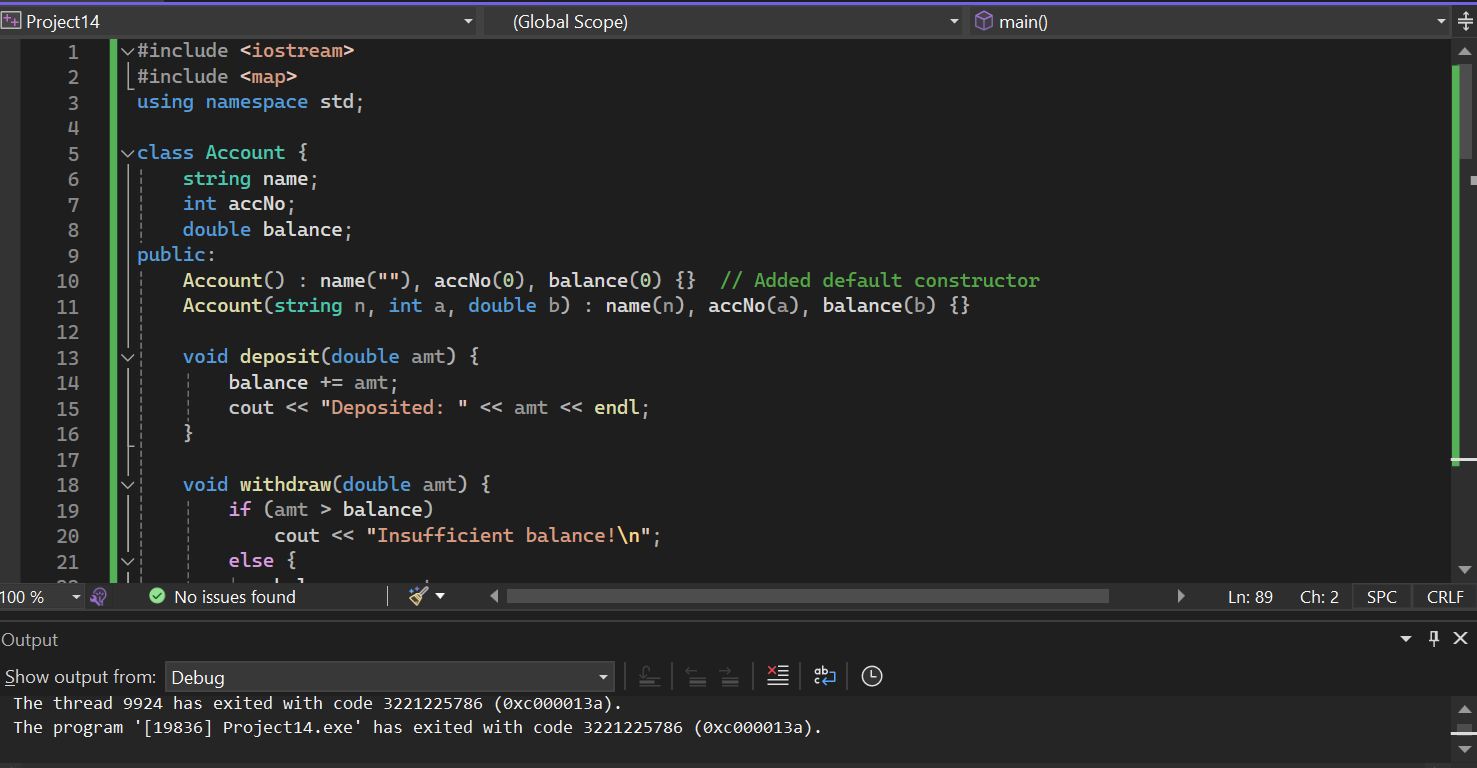
else

cout << "Invalid choice.\n";

}

return 0;

}



Task 3:

#include <iostream>

using namespace std;

#define N 9

// Function to print the Sudoku board

void printBoard(int grid[N][N]) {

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

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

cout << grid[row][col] << " ";

cout << endl;

}

}

// Check if it's safe to place num in grid[row][col]

bool isSafe(int grid[N][N], int row, int col, int num) {

// Check row and column

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

if (grid[row][x] == num || grid[x][col] == num)

return false;

}

// Check 3x3 subgrid

int startRow = row - row % 3;

int startCol = col - col % 3;

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

if (grid[i + startRow][j + startCol] == num)

return false;

return true;

}

// Backtracking solver

bool solveSudoku(int grid[N][N]) {

int row, col;

bool emptyFound = false;

for (row = 0; row < N && !emptyFound; row++) {

for (col = 0; col < N && !emptyFound; col++) {

if (grid[row][col] == 0)

emptyFound = true;

}

}

if (!emptyFound)

return true; // Solved

row--; col--; // Step back to last empty

for (int num = 1; num <= 9; num++) {

if (isSafe(grid, row, col, num)) {

grid[row][col] = num;

if (solveSudoku(grid))

return true;

grid[row][col] = 0; // Backtrack

}

}

return false;

}

int main() {

int grid[N][N] = {

{5,3,0, 0,7,0, 0,0,0},

{6,0,0, 1,9,5, 0,0,0},

{0,9,8, 0,0,0, 0,6,0},

{8,0,0, 0,6,0, 0,0,3},

{4,0,0, 8,0,3, 0,0,1},

{7,0,0, 0,2,0, 0,0,6},

{0,6,0, 0,0,0, 2,8,0},

{0,0,0, 4,1,9, 0,0,5},

{0,0,0, 0,8,0, 0,7,9}

};

if (solveSudoku(grid))

printBoard(grid);

else

cout << "No solution exists.";

return 0;

}

