#include <iostream>

using namespace std;

class Account

{

    private:

        int id = 0;

        double balance = 0.0;

        double annualInterestRate = .045;

        string dateCreated = "3/22/2020";

    public:

        Account()

        {

            balance = 20000;

            id = 1122;

        } // default constructor:

        Account(int id, double balance)

        {

            this -> id = id;

            this -> balance = balance;

        } // regular constructor

        int getId()

        {

            return id;

        }

        double getBal()

        {

            return balance;

        }

        double getMonthlyInterestRate()

        {

            return (annualInterestRate/12.0);

        }

        double getMonthlyInterest()

        {

            return balance \* (annualInterestRate/12.0);

        }

        string getDate()

        {

            return dateCreated;

        }

        void withdraw(double amount)

        {

            balance = balance - amount;

        }

        void deposit(double dep)

        {

            balance = balance + dep;

        }

        void output()

        {

            cout << id << ", " << balance << endl;

        }

};

int main()

{

    //Part 1: instantiate an object Jared using regular constructor

    Account jared(1122, 20000);

    jared.withdraw(2500);

    jared.deposit(3000);

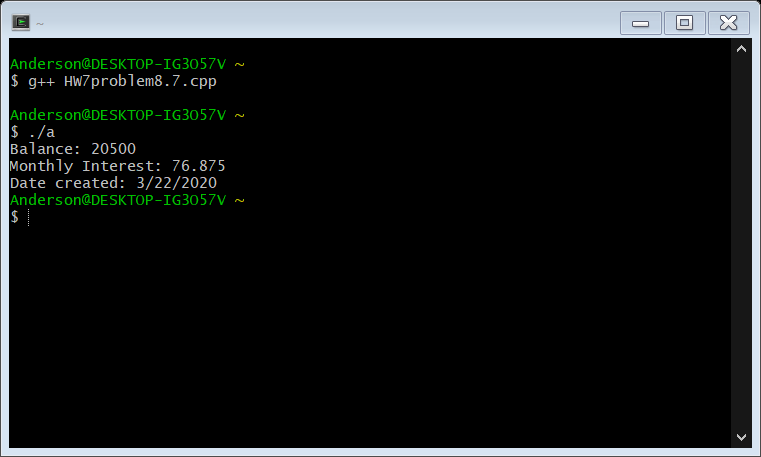
    cout << "Balance: " << jared.getBal() << endl;

    cout << "Monthly Interest: " << jared.getMonthlyInterest() << endl;

    cout << "Date created: " << jared.getDate();

    return 0;

}



#include <iostream>

#include <cmath>

using namespace std;

class QuadraticEquation

{

    private:

        int a, b, c;

    public:

    QuadraticEquation(int a, int b, int c)

    {

        this -> a = a;

        this -> b = b;

        this -> c = c;

    }

    int getA()

    {

        return a;

    }

    int getB()

    {

        return b;

    }

    int getC()

    {

        return c;

    }

    double getDiscriminant()

    {

        return (b\*b-4\*a\*c);

    }

    double getRoot1()

    {

        return ((-b + sqrt(b\*b-4\*a\*c)/(2\*a)));

    }

    double getRoot2()

    {

        return ((-b - sqrt(b\*b-4\*a\*c)/(2\*a)));

    }

};

int main()

{

    int x, y, z;

    cout << "Please enter values for a, b, and c: ";

    cin >> x;

    cin >> y;

    cin >> z;

    QuadraticEquation formula(x, y, z);

    if (formula.getDiscriminant() > 0)

    {

        cout << "Root 1: " << formula.getRoot1() << endl;

        cout << "Root 2: " << formula.getRoot2();

    }

    else if (formula.getDiscriminant() == 0)

    {

        cout << "Root 1: " << formula.getRoot2();

    }

    else

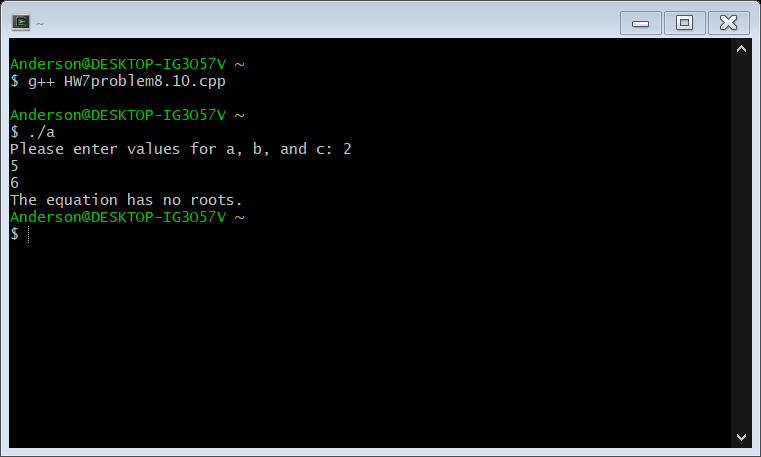
    {

        cout << "The equation has no roots.";

    }

    return 0;

}



#include <iostream>

#include <utility>

using namespace std;

#define pdd pair<double, double>

int flag = 0;

void displayPoint(pdd P)

{

    cout << "(" << P.first << ", " << P.second << ")" << endl;

}

pdd lineIntersection(pdd A, pdd B, pdd C, pdd D)

{

    // Line AB represented as a1x + b1y = c1

    double a1 = B.second - A.second;

    double b1 = A.first - B.first;

    double c1 = a1\*(A.first) + b1\*(A.second);

    // Line CD represented as a2x + b2y = c2

    double a2 = D.second - C.second;

    double b2 = C.first - D.first;

    double c2 = a2\*(C.first)+ b2\*(C.second);

    double determinant = a1\*b2 - a2\*b1;

    if (determinant == 0)

    {

        cout << "The lines are parallel.";

        flag = 1;

    }

    else

    {

        double x = (b2\*c1 - b1\*c2)/determinant;

        double y = (a1\*c2 - a2\*c1)/determinant;

        return make\_pair(x, y);

    }

}

int main()

{

    int l, m, n, o, p, q, x, y;

    cout << "Enter the endpoint of the first line: ";

    cin >> l;

    cin >> m;

    pdd A = make\_pair(l, m);

    cout << "Enter the second endpoint of the first line: ";

    cin >> n;

    cin >> o;

    pdd B = make\_pair(n, o);

    cout << "Enter the endpoint of the second line: ";

    cin >> p;

    cin >> q;

    pdd C = make\_pair(p, q);

    cout << "Enter the second endpoint of the second line: ";

    cin >> x;

    cin >> y;

    pdd D = make\_pair(x, y);

    pdd intersection = lineIntersection(A, B, C, D);

    if (flag == 0)

    {

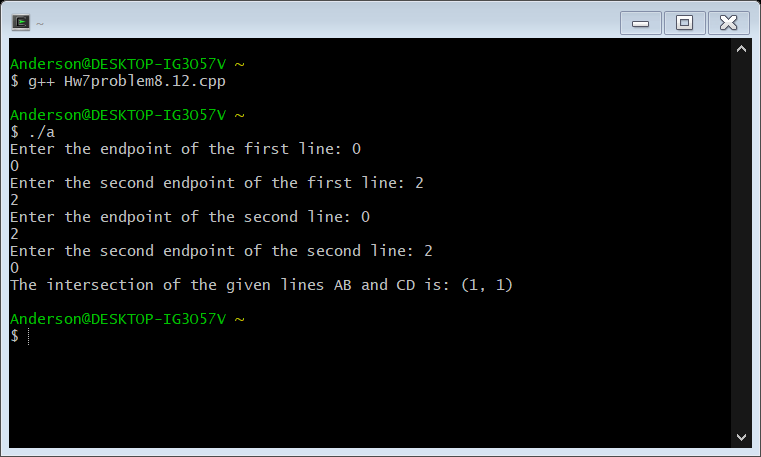
        cout << "The intersection of the given lines AB and CD is: ";

        displayPoint(intersection);

    }

    return 0;

}



#include <iostream>

using namespace std;

class Location

{

    public:

        int row = 0;

        int column = 0;

        double maxValue = 0.0;

    Location(double\*\* a, int row, int column)

    {

        for (int i =0; i < row; i++)

        {

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

            {

                if (a[i][j] > maxValue)

                {

                    this -> row = i;

                    this -> column = j;

                    maxValue = a[i][j];

                }

            }

        }

    }

    double getmaxValue()

    {

        return maxValue;

    }

    int getRow()

    {

        return row;

    }

    int getColumn()

    {

        return column;

    }

};

int main()

{

    int row, column;

    cout << "Please enter the row and column sizes: ";

    cin >> row;

    cin >> column;

    double\*\* twoDimensionalArray = new double\*[row];

    for (int i = 0; i < row; i++)

    {

        twoDimensionalArray[i] = new double[column];

    }

    cout << "Enter " << row\*column << " values for the array: ";

    for (int i = 0; i < row; i++)

    {

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

        {

            cin >> twoDimensionalArray[i][j];

        }

    }

    /\*for (int i = 0; i < row; i++)

    {

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

        {

            cout << twoDimensionalArray[i][j] << " ";

        }

        cout << endl;

    }\*/

    Location derek = Location(twoDimensionalArray, row, column);

    cout << "The largest value is: " << derek.getmaxValue();

    cout << "\nThe location is (" << derek.getRow();

    cout << ", " << derek.getColumn() << ")";

    return 0;

}

