```
Script started on 2024-05-10 09:35:07-05:00 [TERM="xterm" TTY="/dev/pts/0" COLUMNS=
ee43254@ares:~$ pwd
/home/students/ee43254
ee43254@ares:~$ cat algebra.info
Name: Kyle Enkhzul

Class: CSC122-W01



Activity: Polly^Anna

Level: 9.5, 6 (base program), 2.5 (roots function, 1 (evaluate method)



Description:



This program helps you work with algebraic formulas easily. You can

create, manipulate, and evaluate formulas involving any variable.

It also allows one to find the rational roots equations.

ee43254@ares:~$ show-code algebra.cpp


algebra.cpp:
```

```cpp
#include <iostream>
#include <vector>
#include <iomanip>
#include <algorithm>
#include <cmath>

using namespace std;

class AlgebraicFormula {
private:
 vector<double> coefficients;
 char variable;

public:
 // Constructor
 AlgebraicFormula(char var, size_t numTerms) : coefficients(numTerms, 0),
        variable(var) {}

 // Destructor
 ~AlgebraicFormula() {}

 // Overloaded subscript operator
 double& operator[](size_t index) {
  return coefficients[index];
 }

 // Overloaded output operator
 friend ostream& operator<<(ostream& os, const AlgebraicFormula& formula)
  os << fixed << setprecision(2);
   for (size_t i = formula.coefficients.size() - 1;

    if (i != formula.coefficients.size() - 1) {
      os << " + ";
      os << formula.coefficients[i] << formula.variable << "^" << i;
         }
    }
    return os;
 }

 // Overloaded addition operator
 AlgebraicFormula operator+(const AlgebraicFormula& other) const {
  size_t maxSize = max(coefficients.size(), other.coefficients.size());
  AlgebraicFormula result(variable, maxSize);
   for (size_t i = 0; i < maxSize; ++i) {
    double coeff1 = (i < coefficients.size()) ? coefficients[i] : 0;
  double coeff2 = (i < other.coefficients.size()) ? other.coefficients[i]
    result.coefficients[i] = coeff1 + coeff2;
   }
   return result;
 }

 // Overloaded subtraction operator
 AlgebraicFormula operator-(const AlgebraicFormula& other) const {
  size_t maxSize = max(coefficients.size(), other.coefficients.size());
  AlgebraicFormula result(variable, maxSize);
  for (size_t i = 0; i < maxSize; ++i) {
   double coeff1 = (i < coefficients.size()) ? coefficients[i] : 0;
 double coeff2 = (i < other.coefficients.size()) ? other.coefficients[i] :
   result.coefficients[i] = coeff1 - coeff2;
  }
  return result;
 }

 // Overloaded multiplication by a scalar
 AlgebraicFormula operator*(double scalar) const {
  AlgebraicFormula result(variable, coefficients.size());
   for (size_t i = 0; i < coefficients.size(); ++i) {
    result.coefficients[i] = coefficients[i] * scalar;
    }
   return result;
 }

 // Method to get the order of the formula
 size_t getOrder() const {
   return coefficients.size() - 1;
 }

 // Method to get the variable name
 char getVariable() const {
   return variable;
 }

 // Evaluate the formula at a particular value of the variable
 double evaluate(double value) const {
  double result = 0;
  double varPow = 1;
   for (size_t i = 0; i < coefficients.size(); ++i) {
    result += coefficients[i] * varPow;
     varPow *= value;
    }
    return result;
 }

 // Overloaded compound assignment operators
 AlgebraicFormula& operator+=(const AlgebraicFormula& other) {
  *this = *this + other;
 return *this;
 }

 AlgebraicFormula& operator-=(const AlgebraicFormula& other) {
  *this = *this - other;
 return *this;
 }

 AlgebraicFormula& operator*=(double scalar) {
  *this = *this * scalar;
  return *this;
 }

 AlgebraicFormula& operator/=(double scalar) {
  for (size_t i = 0; i < coefficients.size(); ++i) {
   coefficients[i] /= scalar;
   }
  return *this;
 }

// Function to find rational roots
 vector<double> roots() const {
  vector<double> rationalRoots;

  short leadingCoefficient = static_cast<short>(coefficients.back());
  short constantTerm = static_cast<short>(coefficients.front());

  // Find all possible factors of the constant term and leading coefficient
  vector<short> constantFactors;
  vector<short> leadingFactors;

  for(short i = 1; i <= abs(constantTerm); ++i) {
    if (constantTerm % i == 0)
      constantFactors.push_back(i);
    }

  for(short i = 1; i <= abs(leadingCoefficient); ++i) {
    if (leadingCoefficient % i == 0)
      leadingFactors.push_back(i);
    }

  // Test all possible combinations of factors
  const double epsilon = 1e-6;
  for (short constantFactor : constantFactors) {
    for (short leadingFactor : leadingFactors) {
     double root = static_cast<double>(constantFactor) /
                static_cast<double>(leadingFactor);
      if (fabs(evaluate(root)) < epsilon) {
        rationalRoots.push_back(root);
      } else if (fabs(evaluate(-root)) < epsilon) {
        rationalRoots.push_back(-root);
      }
    }
  }

    return rationalRoots;
}

};

int main() {

 AlgebraicFormula f1('x', 3);
 f1[2] = 1;
 f1[1] = 4;
 f1[0] = 4;

 AlgebraicFormula f2('x', 4);
 f2[1] = 5;
 f2[3] = 1;

 AlgebraicFormula f3('y', 3);
 f3[2] = 1;
 f3[1] = 12;
 f3[0] = 36;

 cout << "Given three functions of..." << endl;
 cout << "f1(x) = " << f1 << endl;
 cout << "f2(x) = " << f2 << endl;
 cout << "f3(y) = " << f3 << endl;

 cout << "\nThe sum of f1(x) and f2(x) would be..." << endl;
 AlgebraicFormula f4 = f1 + f2;
 cout << "f4(x) = f1(x) + f2(x) = " << f4 << endl;

 cout << "\nThe difference of f1(x) and f2(x) would be..." << endl;
 AlgebraicFormula f5 = f1 - f2;
 cout << "f5(x) = f1(x) - f2(x) = " << f5 << endl;
```

```cpp

 cout << "\nThe product of multiplying f1(x) by 2.5 would be..." << endl;
 AlgebraicFormula f6 = f1 * 2.5;
 cout << "f6(x) = f1(x) * 2.5 = " << f6 << endl;

 cout << "\nThe highest order of f1(x) is: " << f1.getOrder() << endl;
 cout << "The variable of f1(x) is: " << f1.getVariable() << endl;

 cout << "\nThe highest order of f2(x) is: " << f2.getOrder() << endl;
 cout << "The variable of f2(x) is: " << f2.getVariable() << endl;

 cout << "\nThe highest order of f3(y) is: " << f3.getOrder() << endl;
 cout << "The variable of f3(y) is: " << f3.getVariable() << endl;

 double x_value1 = 2.0;
 cout << "\nEvaluating f1(x) at x = " << x_value1 << " is:" << endl;
 cout << "f1(" << x_value1 << ") = " << f1.evaluate(x_value1) << endl;

 double x_value2 = 5.0;
 cout << "\nEvaluating f2(x) at x = " << x_value2 << " is:" << endl;
 cout << "f2(" << x_value2 << ") = " << f2.evaluate(x_value2) << endl;

 double y_value1 = 3.0;
 cout << "\nEvaluating f3(y) at y = " << y_value1 << " is:" << endl;
 cout << "f3(" << y_value1 << ") = " << f3.evaluate(y_value1) << endl;

 // Finding rational roots of f1
    cout << "\nRational roots of f1(x): ";
    vector<double> f1Roots = f1.roots();
    if (f1Roots.empty()) {
        cout << "None" << endl;
    } else {
        for (double root : f1Roots) {
            cout << root << " ";
        }
        cout << endl;
    }

    // Finding rational roots of f2
    cout << "\nRational roots of f2(x): ";
    vector<double> f2Roots = f2.roots();
    if (f2Roots.empty()) {
        cout << "None" << endl;
    } else {
        for (double root : f2Roots) {
            cout << root << " ";
        }
        cout << endl;
    }

        // Finding rational roots of f2
    cout << "\nRational roots of f3(y): ";
    vector<double> f3Roots = f3.roots();
    if (f3Roots.empty()) {
        cout << "None" << endl;
    } else {
        for (double root : f3Roots) {
            cout << root << " ";
        }
        cout << endl;
    }

    return 0;
}
```

```
ee43254@ares:~$ CPP algebra
algebra.cpp***


ee43254@ares:~$ ./algebra.out
Given three functions of...
f1(x) =  + 4.00x^1 + 4.00x^0
f2(x) =  + 0.00x^2 + 5.00x^1 + 0.00x^0
f3(y) =  + 12.00y^1 + 36.00y^0

The sum of f1(x) and f2(x) would be...
f4(x) = f1(x) + f2(x) =  + 1.00x^2 + 9.00x^1 + 4.00x^0

The difference of f1(x) and f2(x) would be...
f5(x) = f1(x) - f2(x) =  + 1.00x^2 + -1.00x^1 + 4.00x^0

The product of multiplying f1(x) by 2.5 would be...
f6(x) = f1(x) * 2.5 =  + 10.00x^1 + 10.00x^0

The highest order of f1(x) is: 2
The variable of f1(x) is: x

The highest order of f2(x) is: 3
The variable of f2(x) is: x

The highest order of f3(y) is: 2
The variable of f3(y) is: y

Evaluating f1(x) at x = 2.00 is:
f1(2.00) = 16.00

Evaluating f2(x) at x = 5.00 is:
f2(5.00) = 150.00

Evaluating f3(y) at y = 3.00 is:
f3(3.00) = 81.00

Rational roots of f1(x): -2.00

Rational roots of f2(x): None

Rational roots of f3(y): -6.00
ee43254@ares:~$ exit
exit
```

```
Script done on 2024-05-10 09:35:28-05:00 [COMMAND_EXIT_CODE="0"]
```