

```
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:

```
1 #include <iostream>
2 #include <vector>
3 #include <iomanip>
4 #include <algorithm>
5 #include <cmath>
6
7 using namespace std;
8
9 class AlgebraicFormula {
10 private:
11     vector<double> coefficients;
12     char variable;
13
14 public:
15     // Constructor
16     AlgebraicFormula(char var, size_t numTerms) : coefficients(numTerms, 0),
17         variable(var) {}
18
19     // Destructor
20     ~AlgebraicFormula() {}
21
22     // Overloaded subscript operator
```

```
23     double& operator[](size_t index) {
24         return coefficients[index];
25     }
26
27     // Overloaded output operator
28     friend ostream& operator<<(ostream& os, const AlgebraicFormula& formula) {
29         os << fixed << setprecision(2);
30         for (size_t i = formula.coefficients.size() - 1;
31
32             if (i != formula.coefficients.size() - 1) {
33                 os << " + ";
34                 os << formula.coefficients[i] << formula.variable << "^" << i;
35             }
36         }
37         return os;
38     }
39
40     // Overloaded addition operator
41     AlgebraicFormula operator+(const AlgebraicFormula& other) const {
42         size_t maxSize = max(coefficients.size(), other.coefficients.size());
43         AlgebraicFormula result(variable, maxSize);
44         for (size_t i = 0; i < maxSize; ++i) {
45             double coeff1 = (i < coefficients.size()) ? coefficients[i] : 0;
46             double coeff2 = (i < other.coefficients.size()) ? other.coefficients[i] : 0;
47             result.coefficients[i] = coeff1 + coeff2;
48         }
49         return result;
50     }
51
52     // Overloaded subtraction operator
53     AlgebraicFormula operator-(const AlgebraicFormula& other) const {
54         size_t maxSize = max(coefficients.size(), other.coefficients.size());
55         AlgebraicFormula result(variable, maxSize);
56         for (size_t i = 0; i < maxSize; ++i) {
57             double coeff1 = (i < coefficients.size()) ? coefficients[i] : 0;
58             double coeff2 = (i < other.coefficients.size()) ? other.coefficients[i] : 0;
59             result.coefficients[i] = coeff1 - coeff2;
60         }
61         return result;
62     }
63
64     // Overloaded multiplication by a scalar
65     AlgebraicFormula operator*(double scalar) const {
66         AlgebraicFormula result(variable, coefficients.size());
67         for (size_t i = 0; i < coefficients.size(); ++i) {
68             result.coefficients[i] = coefficients[i] * scalar;
69         }
70         return result;
71     }
72
73     // Method to get the order of the formula
74     size_t getOrder() const {
75         return coefficients.size() - 1;
76     }
```

```

77
78 // Method to get the variable name
79 char getVariable() const {
80     return variable;
81 }
82
83 // Evaluate the formula at a particular value of the variable
84 double evaluate(double value) const {
85     double result = 0;
86     double varPow = 1;
87     for (size_t i = 0; i < coefficients.size(); ++i) {
88         result += coefficients[i] * varPow;
89         varPow *= value;
90     }
91     return result;
92 }
93
94 // Overloaded compound assignment operators
95 AlgebraicFormula& operator+=(const AlgebraicFormula& other) {
96     *this = *this + other;
97     return *this;
98 }
99
100 AlgebraicFormula& operator-=(const AlgebraicFormula& other) {
101     *this = *this - other;
102     return *this;
103 }
104
105 AlgebraicFormula& operator*=(double scalar) {
106     *this = *this * scalar;
107     return *this;
108 }
109
110 AlgebraicFormula& operator/=(double scalar) {
111     for (size_t i = 0; i < coefficients.size(); ++i) {
112         coefficients[i] /= scalar;
113     }
114     return *this;
115 }
116
117 // Function to find rational roots
118 vector<double> roots() const {
119     vector<double> rationalRoots;
120
121     short leadingCoefficient = static_cast<short>(coefficients.back());
122     short constantTerm = static_cast<short>(coefficients.front());
123
124     // Find all possible factors of the constant term and leading coefficient
125     vector<short> constantFactors;
126     vector<short> leadingFactors;
127
128     for (short i = 1; i <= abs(constantTerm); ++i) {
129         if (constantTerm % i == 0)
130             constantFactors.push_back(i);

```

```

131     }
132
133     for (short i = 1; i <= abs(leadingCoefficient); ++i) {
134         if (leadingCoefficient % i == 0)
135             leadingFactors.push_back(i);
136     }
137
138     // Test all possible combinations of factors
139     const double epsilon = 1e-6;
140     for (short constantFactor : constantFactors) {
141         for (short leadingFactor : leadingFactors) {
142             double root = static_cast<double>(constantFactor) /
143                           static_cast<double>(leadingFactor);
144             if (fabs(evaluate(root)) < epsilon) {
145                 rationalRoots.push_back(root);
146             } else if (fabs(evaluate(-root)) < epsilon) {
147                 rationalRoots.push_back(-root);
148             }
149         }
150     }
151
152     return rationalRoots;
153 }
154
155 };
156
157 int main() {
158
159     AlgebraicFormula f1('x', 3);
160     f1[2] = 1;
161     f1[1] = 4;
162     f1[0] = 4;
163
164     AlgebraicFormula f2('x', 4);
165     f2[1] = 5;
166     f2[3] = 1;
167
168     AlgebraicFormula f3('y', 3);
169     f3[2] = 1;
170     f3[1] = 12;
171     f3[0] = 36;
172
173     cout << "Given three functions of..." << endl;
174     cout << "f1(x) = " << f1 << endl;
175     cout << "f2(x) = " << f2 << endl;
176     cout << "f3(y) = " << f3 << endl;
177
178     cout << "\nThe sum of f1(x) and f2(x) would be..." << endl;
179     AlgebraicFormula f4 = f1 + f2;
180     cout << "f4(x) = f1(x) + f2(x) = " << f4 << endl;
181
182     cout << "\nThe difference of f1(x) and f2(x) would be..." << endl;
183     AlgebraicFormula f5 = f1 - f2;
184     cout << "f5(x) = f1(x) - f2(x) = " << f5 << endl;

```

```

185
186 cout << "\nThe product of multiplying f1(x) by 2.5 would be..." << endl;
187 AlgebraicFormula f6 = f1 * 2.5;
188 cout << "f6(x) = f1(x) * 2.5 = " << f6 << endl;
189
190 cout << "\nThe highest order of f1(x) is: " << f1.getOrder() << endl;
191 cout << "The variable of f1(x) is: " << f1.getVariable() << endl;
192
193 cout << "\nThe highest order of f2(x) is: " << f2.getOrder() << endl;
194 cout << "The variable of f2(x) is: " << f2.getVariable() << endl;
195
196 cout << "\nThe highest order of f3(y) is: " << f3.getOrder() << endl;
197 cout << "The variable of f3(y) is: " << f3.getVariable() << endl;
198
199 double x_value1 = 2.0;
200 cout << "\nEvaluating f1(x) at x = " << x_value1 << " is:" << endl;
201 cout << "f1(" << x_value1 << ") = " << f1.evaluate(x_value1) << endl;
202
203 double x_value2 = 5.0;
204 cout << "\nEvaluating f2(x) at x = " << x_value2 << " is:" << endl;
205 cout << "f2(" << x_value2 << ") = " << f2.evaluate(x_value2) << endl;
206
207 double y_value1 = 3.0;
208 cout << "\nEvaluating f3(y) at y = " << y_value1 << " is:" << endl;
209 cout << "f3(" << y_value1 << ") = " << f3.evaluate(y_value1) << endl;
210
211 // Finding rational roots of f1
212 cout << "\nRational roots of f1(x): ";
213 vector<double> f1Roots = f1.roots();
214 if (f1Roots.empty()) {
215     cout << "None" << endl;
216 } else {
217     for (double root : f1Roots) {
218         cout << root << " ";
219     }
220     cout << endl;
221 }
222
223 // Finding rational roots of f2
224 cout << "\nRational roots of f2(x): ";
225 vector<double> f2Roots = f2.roots();
226 if (f2Roots.empty()) {
227     cout << "None" << endl;
228 } else {
229     for (double root : f2Roots) {
230         cout << root << " ";
231     }
232     cout << endl;
233 }
234
235 // Finding rational roots of f3
236 cout << "\nRational roots of f3(y): ";
237 vector<double> f3Roots = f3.roots();
238 if (f3Roots.empty()) {

```

```

239     cout << "None" << endl;
240 } else {
241     for (double root : f3Roots) {
242         cout << root << " ";
243     }
244     cout << endl;
245 }
246
247 return 0;
248 }

```

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

ee43254@ares:~\$./algebra.out

Given three functions of...

$f_1(x) = + 4.00x^1 + 4.00x^0$

$f_2(x) = + 0.00x^2 + 5.00x^1 + 0.00x^0$

$f_3(y) = + 12.00y^1 + 36.00y^0$

The sum of $f_1(x)$ and $f_2(x)$ would be...

$f_4(x) = f_1(x) + f_2(x) = + 1.00x^2 + 9.00x^1 + 4.00x^0$

The difference of $f_1(x)$ and $f_2(x)$ would be...

$f_5(x) = f_1(x) - f_2(x) = + 1.00x^2 + -1.00x^1 + 4.00x^0$

The product of multiplying $f_1(x)$ by 2.5 would be...

$f_6(x) = f_1(x) * 2.5 = + 10.00x^1 + 10.00x^0$

The highest order of $f_1(x)$ is: 2

The variable of $f_1(x)$ is: x

The highest order of $f_2(x)$ is: 3

The variable of $f_2(x)$ is: x

The highest order of $f_3(y)$ is: 2

The variable of $f_3(y)$ is: y

Evaluating $f_1(x)$ at $x = 2.00$ is:

$f_1(2.00) = 16.00$

Evaluating $f_2(x)$ at $x = 5.00$ is:

$f_2(5.00) = 150.00$

Evaluating $f_3(y)$ at $y = 3.00$ is:

$f_3(3.00) = 81.00$

Rational roots of $f_1(x)$: -2.00

Rational roots of $f_2(x)$: None

Rational roots of $f_3(y)$: -6.00

ee43254@ares:~\$ exit

exit

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