

```
1  #include <algorithm>
2  #include <numeric>           // To use GCD
3  #include <cmath>
4  #include <iostream>
5  #include "Assignment6.h"
6
7  using namespace std;
8
9  //////////////////////////////////////
10 //          METHODS/FUNCTIONS DEFINITIONS          //
11 //////////////////////////////////////
12
13 // Default constructor - To set an object to a known state
14 fraction::fraction(void) {
15     n = 0;
16     d = 1;
17 }
18
19 // Basically another constructor...
20 void fraction::setFraction(int n, int d) {
21     this->n = n;
22     this->d = d;
23 }
24
25
26 void fraction::setFraction(fraction f) {
27     this->n = f.n;
28     this->d = f.d;
29 }
30
31
32 // From here on, notice that it makes no sense to simplify if
33 // either the numerator or the denominator are equal to 1 or if
34 // the numerator is equal to 0.
35
36 //*****
37
38 void fraction::add(fraction f) {
39     n = (f.d * n + f.n * d);
40     d *= f.d;
41     if (n != 0 && n != 1 && d != 1)
42         simplify();
43 }
44
45
46 //*****
47 void fraction::mult(fraction f) {
48     n *= f.n;
49     d *= f.d;
50     if (n != 0 && n != 1 && d != 1)
51         simplify();
52 }
53
54
55
56
57
58
```

```
59
60
61 //*****
62 void fraction::div(fraction f) {
63     n = n * f.d;
64     d = d * f.n;
65     if (n != 0 && n != 1 && d != 1)
66         simplify();
67 }
68
69
70 //*****
71 void fraction::display(void) {
72     if (n == 1 && d == 1)
73         cout << 1 << endl;
74     else if (n == 0)
75         cout << 0 << endl;
76     else if (d == 1)
77         cout << n << endl;
78     else
79         cout << n << " / " << d << endl;
80 }
81
82
83 //*****
84 // NB: To use std::gcd be sure that the IDE is set to C++17
85 // for the project. Because of this I originally wrote my own
86 // functions (that I left below).
87
88 void fraction::simplify() {
89     int m;
90     // Simple cases first because computation is elementary
91     if (n == d) {
92         n = 1;
93         d = 1;
94     }
95     else if ((d % n == 0) && (n < d)) {
96         d /= n;
97         n = 1;
98     }
99     else if ((n % d == 0) && (n > d)) {
100         n /= d;
101         d = 1;
102     }
103     else {
104         m = (n > d) ? std::gcd(n, d) : std::gcd(d, n);
105
106         // m = naiveGCD()
107         // m = (n > d) ? euclidianGCD(n, d) : euclidianGCD(d, n);
108         // m = (n > d) ? binaryGCD(n, d) : binaryGCD(d, n);
109         n /= m;
110         d /= m;
111     }
112 }
113
114
115
116
```

```
117 //*****
118 // Naive solution
119
120 int fraction::naiveGCD() {
121     int gcd;
122     for (int i = 1; i <= n && i <= d; i++)
123         if (n % i == 0 && d % i == 0)
124             gcd = i;
125     return gcd;
126 }
127
128
129 // Euclidean Algorithm for GCD - Recursive
130 // Should be very efficient for small numbers though I'm not sure
131 // how small is "small"...
132
133 int fraction::euclidianGCD(int a, int b) {
134     if (a == 0) // probably unnecessary
135         return b;
136     else if (b == 0)
137         return a;
138     else {
139         return euclidianGCD(b, a % b);
140     }
141 }
142
143
144 // Binary GCD - Recursive
145 // It should be even more efficient than Euclidian.
146
147 int fraction::binaryGCD(int a, int b) {
148     if (a == 0) // probably unnecessary
149         return b;
150     if (b == 0)
151         return a;
152     // a is even
153     if (a % 2 == 0)
154
155         if (b % 2 == 0)
156             // both are even
157             return 2 * binaryGCD(a / 2, b / 2);
158         else
159             // a is even and b is odd
160             return binaryGCD(a / 2, b);
161     // a is odd
162     else
163         // a is odd and b is even
164         if (b % 2 == 0)
165             return binaryGCD(a, b / 2);
166
167         else
168             // both are odd
169             return binaryGCD(abs(a - b) / 2, b);
170
171     // in case of mistakes... and to make some compilers happy
172     return 0;
173 }
```