

```
1  #include <iostream>
2  #include <sstream>
3  #include <string>
4  #include <iomanip>
5  #include "Assignment6.h"
6
7
8  // Uncomment this to use in_stream.ignore
9  // auto max_size = std::numeric_limits<std::streamsize>::max();
10
11
12 void calculate(std::stringstream & in_stream, fraction & result) {
13     char c;
14     int numerator, denominator;
15     fraction f;
16     std::string ops;
17
18     while (!in_stream.eof()) {
19         // Read numerator
20         in_stream >> numerator;
21
22         // Drop/skip the fraction sign '/'
23         // in_stream.ignore(max_size, '/');
24         in_stream >> c;
25
26         // Read denominator
27         in_stream >> denominator;
28
29         // Create the fraction
30         f.setFraction(numerator, denominator);
31
32         // Check if there's an operator saved
33         if (ops == "+")
34             result.add(f);
35         else if (ops == "*")
36             result.mult(f);
37         else if (ops == "div")
38             result.div(f);
39         else if (ops.empty())
40             result.setFraction(f);
41         else {
42             std::cout << "error" << std::endl;
43             return;
44         }
45
46         // Ignore all white spaces until the next "non-white"
47         // character that can be peeked
48         // NB In some compilers this may not be necessary
49         in_stream >> std::ws;
50
51         // Read operator
52         if (!in_stream.eof())
53             if (in_stream.peek() == '+' || in_stream.peek() == '*')
54                 in_stream >> std::setw(1) >> ops;
55         if (in_stream.peek() == 'd')
56             in_stream >> std::setw(3) >> ops;
57     }
58 }
```

```
59
60
61
62
63 ///////////////////////////////////////////////////////////////////
64
65
66
67
68 // How it works:
69 // It reads a fraction and checks if there is an operator saved.
70 // If there's none, it means that the first operand was read and
71 // the result is just set to the fraction.
72 // If there's an operator saved, the fraction is updated by the
73 // corresponding operation (this in-place behaviour is required
74 // by the assignment).
75 //
76 // Example:
77 //
78 // 2 / 3 + 3 / 7
79 //
80 // 1. Read 2/3 (loop 1)
81 // 2. There's no operator saved: result = 2/3 (loop 1)
82 // 3. Save '+' (loop 1)
83 // 4. Read 3/7 (loop 2)
84 // 5. There's an operator '+' saved (loop 2)
85 // 6. result = 2/3 + 3/7 = 23/21
86
87 ///////////////////////////////////////////////////////////////////
88
89 // Comments:
90
91 // * One way to drop '/' is to import it into a character and then
92 // forget about it.
93 // Though reading '/' and not using it is not very elegant, the
94 // mechanism to skip data in a stringstream is overly complicated
95 // for just one single character. Also it requires to create an instance
96 // of a class that is not used for anything else and might take more space
97 // than a simple char.
98 // I'm not sure what happens behind the scenes and what is actually
99 // more efficient...
100
101 // Because of the different length of operators that have to be checked,
102 // the stream needs to be peeked to know how many characters to read into 'ops'
103
104
105
106
107
108
109
110
111
112
113
114
115
116
```

```
117 ///////////////////////////////////////////////////////////////////
118 // The same function can operate recursively.
119 // It works in a way that's easier to understand:
120 // - Read a fraction
121 // - Read an operator
122 // - Call the function recursively in the context of the operation specified
123 ///////////////////////////////////////////////////////////////////
124
125
126 fraction calculateRecursive(std::stringstream & in_stream) {
127     fraction f;
128     std::string ops;
129     char c;
130     int numerator, denominator;
131
132     if (!in_stream.eof()) {
133         in_stream >> numerator;
134         in_stream >> c;
135         in_stream >> denominator;
136
137         f.setFraction(numerator, denominator);
138
139         in_stream >> std::ws;
140
141         if (!in_stream.eof())
142             if (in_stream.peek() == '+' || in_stream.peek() == '*')
143                 in_stream >> std::setw(1) >> ops;
144             if (in_stream.peek() == 'd')
145                 in_stream >> std::setw(3) >> ops;
146
147         if (ops == "+")
148             f.add(calculateRecursive(in_stream));
149         else if (ops == "*")
150             f.mult(calculateRecursive(in_stream));
151         else if (ops == "div")
152             f.div(calculateRecursive(in_stream));
153     }
154     return f;
155 }
156
157
158
159 ///////////////////////////////////////////////////////////////////
160
161 // I don't like either solution... They're a bit too "twisted".
```