```cpp
#include <iostream>
#include <sstream>
#include <string>
#include <iomanip>
#include "Assignment6.h"


// Uncomment this to use in_stream.ignore
// auto max_size = std::numeric_limits<std::streamsize>::max();


void calculate(std::stringstream & in_stream, fraction & result) {
    char c;
    int numerator, denominator;
    fraction f;
    std::string ops;

    while (!in_stream.eof()) {
        // Read numerator
        in_stream >> numerator;

        // Drop/skip the fraction sign '/'
        // in_stream.ignore(max_size, '/');
        in_stream >> c;

        // Read denominator
        in_stream >> denominator;

        // Create the fraction
        f.setFraction(numerator, denominator);

        // Check if there's an operator saved
        if (ops == "+")
            result.add(f);
        else if (ops == "*")
            result.mult(f);
        else if (ops == "div")
            result.div(f);
        else if (ops.empty())
            result.setFraction(f);
        else {
            std::cout << "error" << std::endl;
            return;
        }

        // Ignore all white spaces until the next "non-white"
        // character that can be peeked
        // NB In some compilers this may not be necessary
        in_stream >> std::ws;

        // Read operator
        if (!in_stream.eof())
            if (in_stream.peek() == '+' || in_stream.peek() == '*')
                in_stream >> std::setw(1) >> ops;
        if (in_stream.peek() == 'd')
            in_stream >> std::setw(3) >> ops;
    }
}
```

```
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  // corrisponding 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. Though this isn't very elegant, the
93  // mechanism to skip data in a stringstream is overly complicated
94  // for just one single character. Also it requires to create an instance
95  // of a class that is not used for anything else and might take more space
96  // than a simple char variable.
97  // I'm not sure what happens behind the scenes and what is actually
98  // more efficient...
99
100 // Because of the different lenght of operators that have to be checked,
101 // the streamm needs to be peeked to know how many characters to read into 'ops'
102 // In the case of + and * only one character is read from in_stream using std::setw(1)
103 // In the case of division the operator is "div" so 3 characters are read (std::setw
      (3))
104
105
106
107
108
109
110
111
112
113
114
115
```

```cpp


////////////////////////////////////////////////////////////////////////
// The same function can operate recursively.
// It works in a way that's easier to understand:
// - Read a fraction
// - Read an operator
// - Call the function recursively in the context of the operation specified
////////////////////////////////////////////////////////////////////////

// EDIT: Actually, it doesn't do the same in general!
//       The function 'calculate' associates operands left to right (with no
//       operand precedence) as required in the assignment.
//       This recursive solution reads the first operand and then associates
//       the rest of the operands right to left after applying the last
//       operator read to the first and last operands...
//       If the operators are all '+' or '*', it doesn't matter, but with
//       'div' and mixed operators the result is wrong.
//       I'll leave it as a starting point if one wants to modify it to make
//       it work.


fraction calculateRecursive(std::stringstream & in_stream) {
    fraction f;
    std::string ops;
    char c;
    int numerator, denominator;

    if (!in_stream.eof()) {
        // Read a fraction
        in_stream >> numerator;
        in_stream >> c;
        in_stream >> denominator;

        f.setFraction(numerator, denominator);

        // Skip to the operator
        in_stream >> std::ws;

        // Read Operator
        if (!in_stream.eof())
            if (in_stream.peek() == '+' || in_stream.peek() == '*')
                in_stream >> std::setw(1) >> ops;
            if (in_stream.peek() == 'd')
                in_stream >> std::setw(3) >> ops;

        if (ops == "+")
            f.add(calculateRecursive(in_stream));
        else if (ops == "*")
            f.mult(calculateRecursive(in_stream));
        else if (ops == "div")
            f.div(calculateRecursive(in_stream));
    }
    return f;
}

// I don't like either solution... They're a bit too "twisted".

```