Using Logo to Solve Sommerville's Algorithm for Adding Mixed Numbers

Ron Taylor

Ron Taylor holds a B.Sc. in animal biology and a B.Ed. in secondary science. He taught Grades 1 through 12 in Rocky View School Division No. 41. He is currently studying for a master's degree in education at the University of Lethbridge and is interested in the phenomenology and pedagogy of play.

One of the important aspects of problem solving that we teach students is to look back at a successful solution and determine if the problem can be solved in another manner. What follows is just such an attempt. In this case, the essential algorithm, developed by Francis Sommerville (1987), has not been changed. The intention here is to show how the algorithm can be implemented in another language, Logo.

The Logo language has several advantages over traditional versions of microcomputer BASIC, among them extensibility and meaningful variable names. By extensibility, I mean that the language allows students to incorporate task names (see Figure 1) directly into the program. Using meaningful variable names helps students to avoid the confusion that sometimes results when many variables are used. The result is a longer program, but one that, with practice, may help the student to better understand the program or to develop alternate solutions.

Translating a program from BASIC into Logo may seem sacrilegious to many students and teachers familiar with the Logo language. Logo offers the particularly strong problem solving tool of recursion to emulate the loops used in Sommerville's program. The program does not fully exploit the power of the recursive loop but does retain Sommerville's original algorithm. Excellent articles on the use of the recursive loop can be found in the journal *The Computing Teacher*. Cathcart (1987) has recently published an article discussing the use of the recursive loop to generate factors. Readers may wish to develop Logo programs to add mixed numbers that use the Logo language to full advantage.

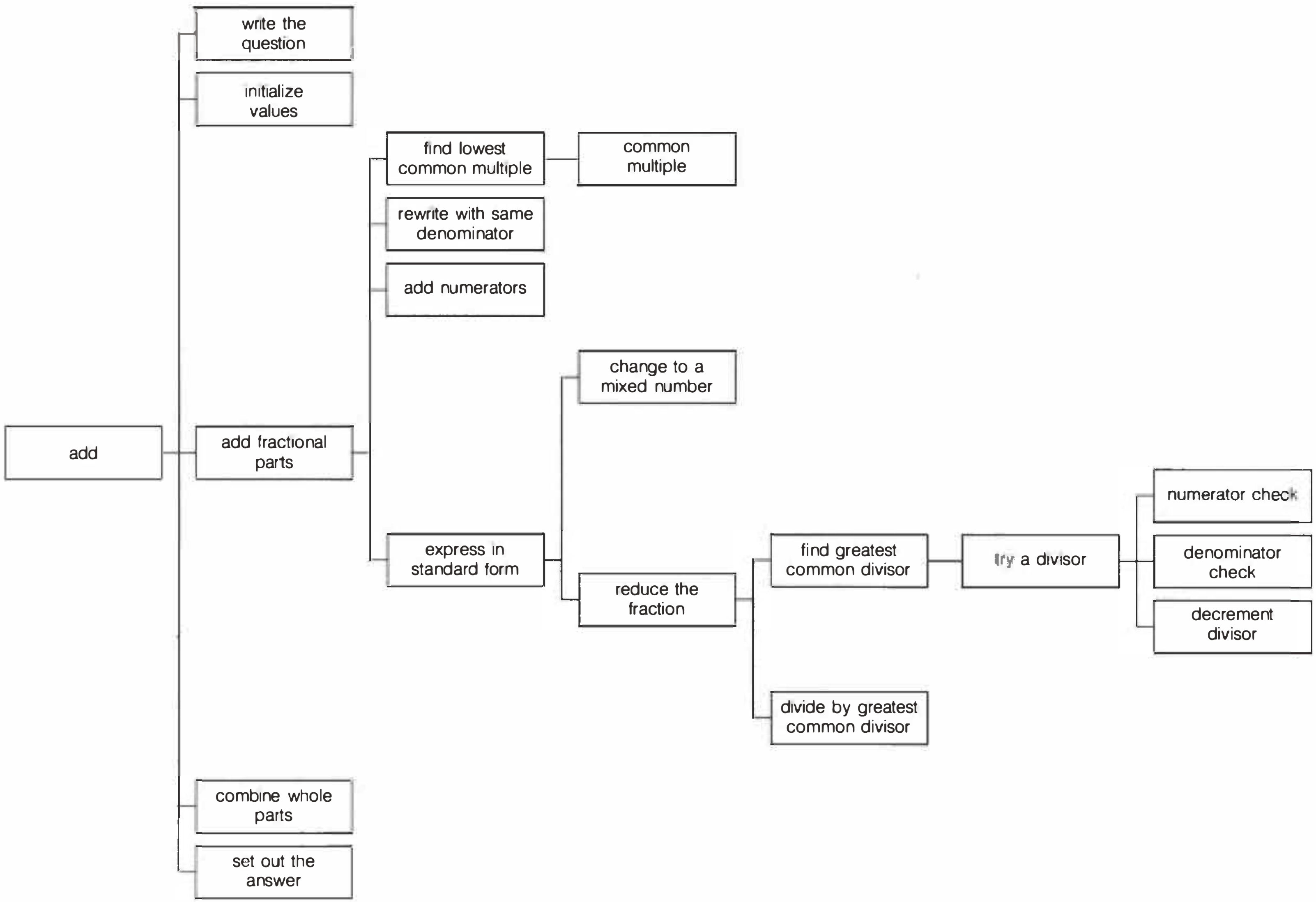
Logo Program

```
TO add :whole1 :numerator1 :denominator1 :whole2 :numerator2 :denominator2
        write.the.question
        initialize.values
        add.fractional.parts
        combine.whole.parts
        set.out.the.answer
END

TO write.the.question
        PRINT (SENTENCE :whole1 :numerator1 [/] :denominator1 [+] :whole2
            :numerator2 [/] :denominator2)
END
```

```
TO initialize.values
        MAKE "sum.of.numerators 0
        MAKE "multiplier 0
        MAKE "equiv.numerator1 0
        MAKE "equiv.numerator2 0
        MAKE "lowest.common.denominator 0
        MAKE "total.whole 0
        MAKE "reduced.denominator 0
        MAKE "check 0
        MAKE "derived.whole.part 0
        MAKE "reduced.numerator 0
        MAKE "not.reduced.numerator 0
        MAKE "test.divisor 0
END

TO add.fractional.parts
        find.lowest.common.multiple :denominator1 :denominator2
        rewrite.with.same.denominator
        add.numerators
        express.in.standard.form
END

TO find.lowest.common.multiple :denominator1 :denominator2
        common.multiple :denominator1 :denominator2
        MAKE "lowest.common.denominator :check
END

TO common.multiple :denominator1 :denominator2
        MAKE "multiplier :multiplier + 1
        MAKE "check :denominator1 * :multiplier
        TEST 0 = REMAINDER :check :denominator2
        IFFALSE [common.multiple :denominator1 :denominator2]
        STOP
END

TO rewrite.with.same.denominator
        MAKE "equiv.numerator1 :lowest.common.denominator / :denominator1 * :numerator1
        MAKE "equiv.numerator2 :lowest.common.denominator / :denominator2 * :numerator2
END

TO add.numerators
        MAKE "sum.of.numerators :equiv.numerator1 + :equiv.numerator2
END

TO express.in.standard.form
        change.to.a.mixed.number
        reduce.the.fraction
END

TO change.to.a.mixed.number
        MAKE "derived.whole.part INT :sum.of.numerators /
            :lowest.common.denominator
        MAKE "not.reduced.numerator :sum.of.numerators - :derived.whole.part *
```

```
                :lowest.common.denominator
END

TO reduce.the.fraction
        find.the.greatest.common.divisor
        divide.by.the.greatest.common.divisor
END

TO find.the.greatest.common.divisor
        MAKE " test.divisor :lowest.common.denominator
        try.a.divisor
        MAKE." greatest.common.divisor :test.divisor
END

TO try.a.divisor
        IF :test.divisor = 1 [STOP]
        IF NOT (AND numerator.check = "true denominator.check = "true)
            [decrement.divisor try.a.divisor]
        STOP
END

TO numerator.check
        TEST 0 = REMAINDER :not.reduced.numerator :test.divisor
        IFTRUE [ OUTPUT "true]
        OUTPUT [" false]
END

TO denominator.check
        TEST 0 = REMAINDER :lowest.common.denominator :test.divisor
        IFTRUE [ OUTPUT "true]
        OUTPUT [" false]
END

TO decrement.divisor
        MAKE "test.divisor:test.divisor - 1
END

TO divide.by.the.greatest.common.divisor
        MAKE "reduced.numerator. INT :not.reduced.numerator /
            :greatest.common.divisor
        MAKE "reduce.denominator INT :lowest.common.denominator /
            :greatest.common.divisor
END

TO combine.whole.parts
        MAKE "total.whole :whole1 + :whole2 + :derived.whole.part
END

TO set.out.the.answer
        PRINT (SENTENCE :total.whole :reduce.numerator [/] :reduced.denominator)
END
```

Figure 1. Warnier/Orr Diagram of Task Names



References

Cathcart, W.G. "Generating Factors." *Computers in Education*, February 1987, pp. 28–29.

Sommerville, F. "Programming: A Subset of Problem Solving." *delta-K*.