This assignment is an elaboration of the in-class lab on generators. The goal is to extend the GENERATORS language with three new kinds of expressions:

(A) A not-expression has the syntax

```
not <expression>
```

It fails when its subexpression generates at least one value and succeeds, yielding the value 143 (once), when its subexpression fails. The subexpression of a not-expression should be bounded. (In other words, it should not be resumed when the not-expression fails!)

(B) A while-expression has the syntax

```
while <expression> do <expression>
```

When a while-expression is evaluated, its first subexpression is evaluated first. If this subexpression succeeds, the second subexpression of the while-expression is evaluated, and then the first subexpression is evaluated again (from the beginning — re-evaluated, not resumed). If and when the first subexpression fails, the entire while-expression fails.

Note that both subexpressions of a while-expression are bounded: Goal-directed evaluation can occur within either subexpression, but when either of them fails, the other is not resumed.

(C) A cycle-expression has the syntax

```
cycle <expression>
```

When a cycle-expression is evaluated, it evaluates its subexpression and yields the value of the subexpression, but fails if the subexpression fails. If resumed after yielding its first value, it resumes the subexpression and continues to yield the successive values of the subexpression. However, when the subexpression can yield no more values, the cycle-expression evaluates it from the beginning again and yields any values that the subexpression can yield from this renewed evaluation.

For example, the expression `cycle from 1 to 3` has the value 1 when first evaluated; on successive resumptions, it yields the values 2, 3, 1, 2, 3, 1, 2, 3, and so on indefinitely. The expression `cycle 1` generates an endless stream of 1s.

Note, however, that it is possible for a cycle-expression to generate only finitely many values. For instance, the program

```
let i = 5
in every cycle begin
    print(i);
    set i = -(i, 1);
    not zero?(i)
end
```

will stop after it has printed 5, 4, 3, 2, and 1.
This assignment will be due on Wednesday, November 18. Please submit your solution in hard copy at the beginning of that day’s class session.