After a couple of examples, the other students caught on. The teacher then goes on to help the students develop the rest of the program. A description follows.

The discovered formula stated in "Logoese."

```
Make "days 10
make "wages1 (:days - 1) * 2 + 50
print :wages1
```

The computer prints 68 confirming Steve's previous calculation. If we want to find out how much one gets on the last day (22), we change the first line to `make "days 22`. The formula is applied and `wages1` becomes $2100. (This appears to be an impressive amount.)

A similar formula can be written for the second payment plan. The formula should deliver 1¢ for the first day and will indicate what is earned on any given day.

The value of wages2 gets multiplied by 2 and this becomes the new value of wages2. This gets repeated 4 times. The value of wages2 which started as .01 becomes .02, .04, .08, and finally .16

```
Make "days 5
make "wages2 .01
repeat 4 [make "wages2 :wages2 *
2]
```

So on the fifth day, you would earn $.16. We also need a formula to see what our total earnings are for the month. For plan #1:

```
make "total.wages.plan1 0
make "wages.plan1 (:days-1)*2+50
make "total.wages.plan1 :total.wages.plan1
+ :wages.plan1
```

The new `total.wages.plan1` is the old total (0) plus day1's earnings (50) which now equals 50. The same is done with plan #2. The initial total is set to 0.

```
make "total.wages.plan2 0
```

On day 1 `wages.plan2` is set to 1.

```
if :day = 1 [make "wages.plan2 1]
```

On subsequent days the repeat formula is used to determine the wages for a given day.

```
if :days > 1 [repeat (:days - 1)
     [make "wages.plan2 :wages.plan2 * 2]]
```

The new `total.wages.plan1` is the old total (0) plus day one's earnings (.01) which now equals .01.

```
make "total.wages.plan2 :total.wages.plan2 +
:wages.plan2
```

the results of the day's transaction are printed out for all to see.

```
pr (se :wages.plan1 :wages.plan2
:total.wages.plan1 :total.wages.plan2)
```

These instructions are next put inside a procedure with a recursive call so the calculations can be performed for a given number of days.

```
to loop :count :days
if (:count > :days) [ stop]
make "wages.plan1 (:count - 1) * 2 + 50
make "total.wages.plan1 :total.wages.plan1 +
:wages.plan1
if :count = 1 [make "wages.plan2 .01]
if :count > 1 [make "wages.plan2 :wages.plan2
* 2]
make "total.wages.plan2 :total.wages.plan2 +
:wages.plan2
(pr :count :wages.plan1 :wages.plan2
:total.wages.plan1 :total.wages.plan2
(:total.wages.plan1 - :total.wages.plan2))
loop :count + 1 :days
end
```

Finally, a superprocedure `report` is written to initialize variables and limit the number of inputs to just one. The procedure becomes this:

Continued on next page