PERFORM S2 THRU E2 VARYING K FROM 1 BY 1 UNTIL K IS GREATER THAN 1000.

An Ode to Insanity by Jason Nguyen

PERFORM VS. GOTO

- **PERFORM** is like a function call
- **GO TO** is an unconditional jump

PERFORM

```
perform para1.
stop run.

para1.
    display “hello”.

para2.
    display “world”.
```

This is the main part of the program. Think of this part as the body of the **main()**.

PERFORM

```
perform para1.
stop run.

para1.
    display “hello”.

para2.
    display “world”.
```

The first instruction of this program is to **perform para1**.

PERFORM

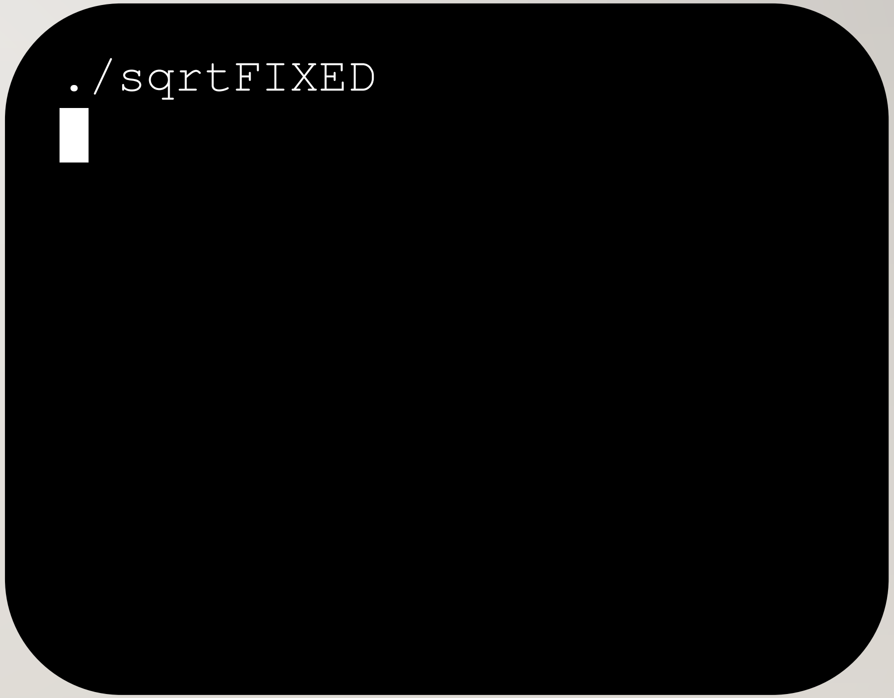
```
perform para1.
stop run.

para1.
    display “hello”.

para2.
    display “world”.
```

It’s right here!

PERFORM

```
perform para1.
stop run.

para1.
    display “hello”.

para2.
    display “world”.
```

It’s right here!

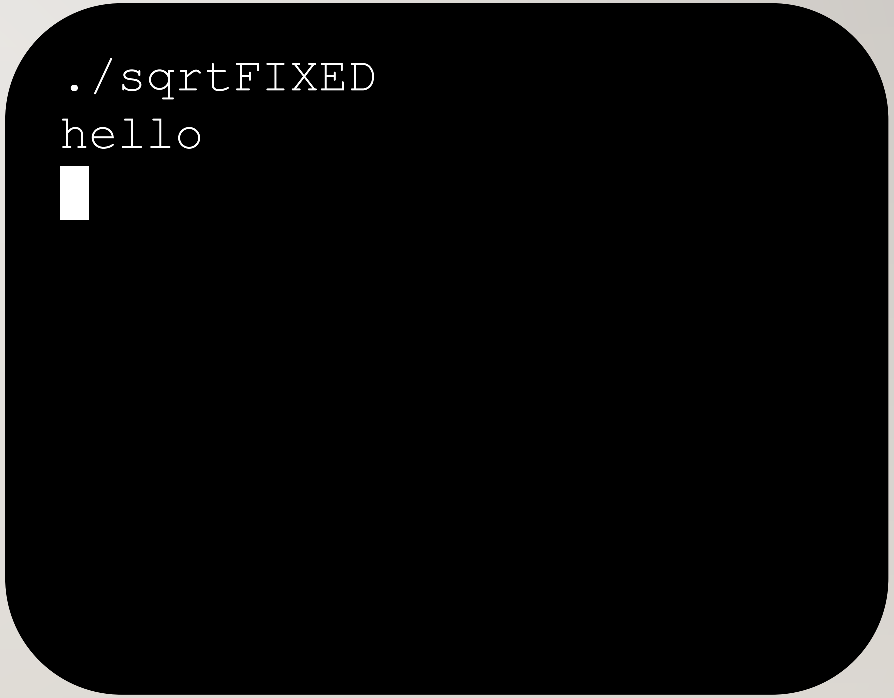


```
./sqrtFIXED
█
```

PERFORM

```
perform para1.
stop run.

para1.
    display “hello”.

para2.
    display “world”.
```

It’s right here!

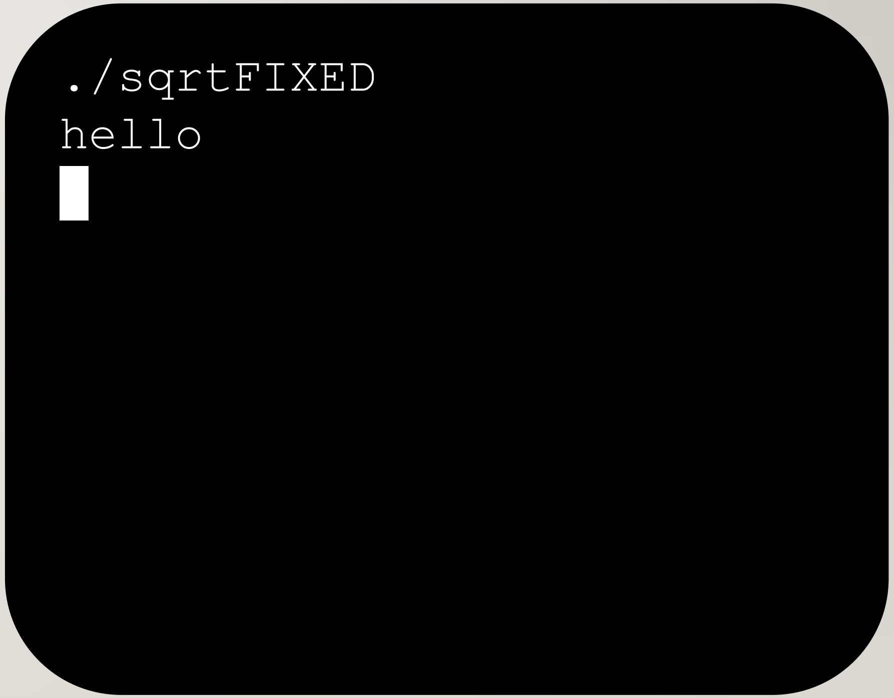


```
./sqrtFIXED
hello
█
```

PERFORM

```
perform para1.
stop run.

para1.
    display “hello”.

para2.
    display “world”.
```

Paragraph is done.

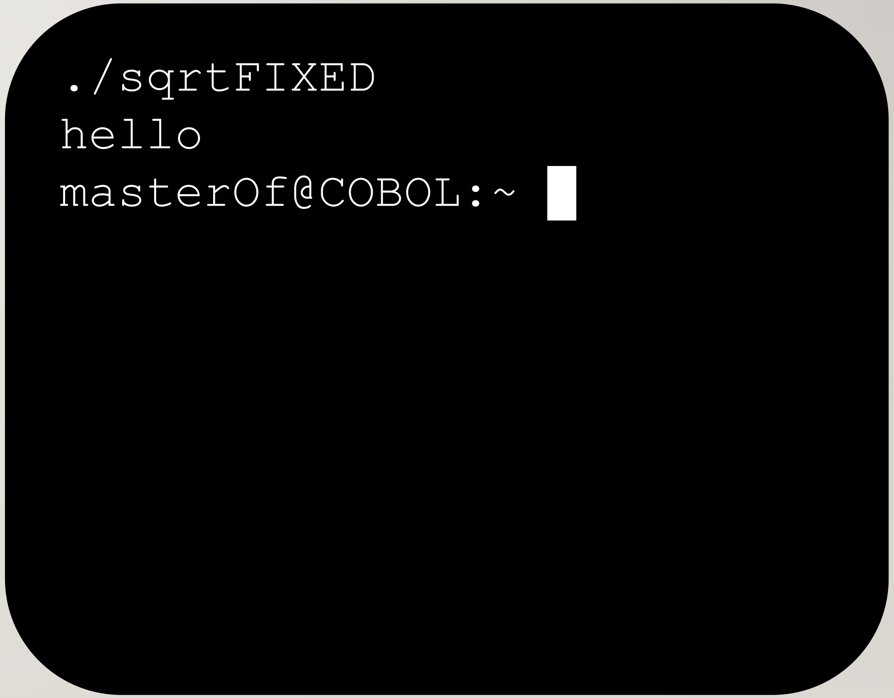


```
./sqrtFIXED
hello
█
```

PERFORM

```
perform para1.
stop run.

para1.
    display “hello”.

para2.
    display “world”.
```

Program terminates



```
./sqrtFIXED
hello
masterOf@COBOL:~ █
```

GO TO

```
GO TO para1.
stop run.

para1.
    display “hello”.

para2.
    display “world”.
```

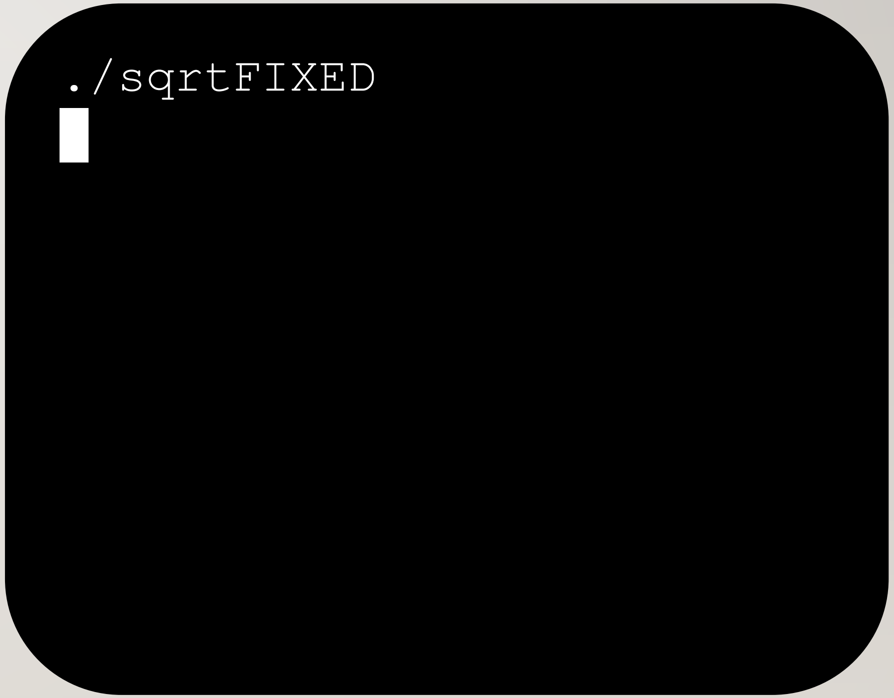
Let’s use an example with **GO TO**.

```
./sqrtFIXED
```

GO TO

```
GO TO para1.
stop run.

para1.
    display “hello”.

para2.
    display “world”.
```

GO TO means **GO TO AND NEVER COME BACK.**

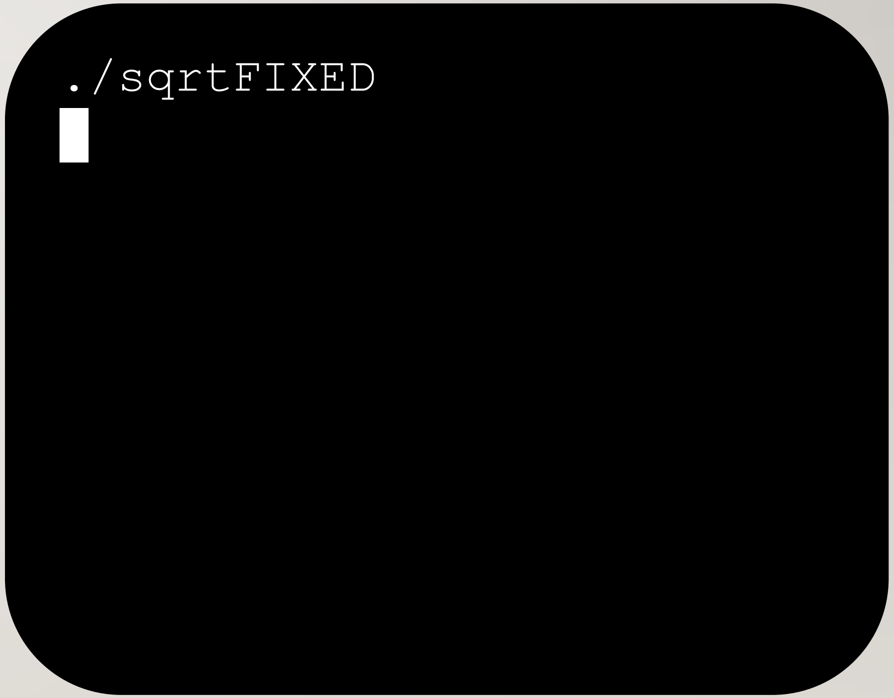


```
./sqrtFIXED
█
```

GO TO

```
GO TO para1.
stop run.

para1.
    display “hello”.

para2.
    display “world”.
```

GO TO means **GO TO AND NEVER COME BACK.**

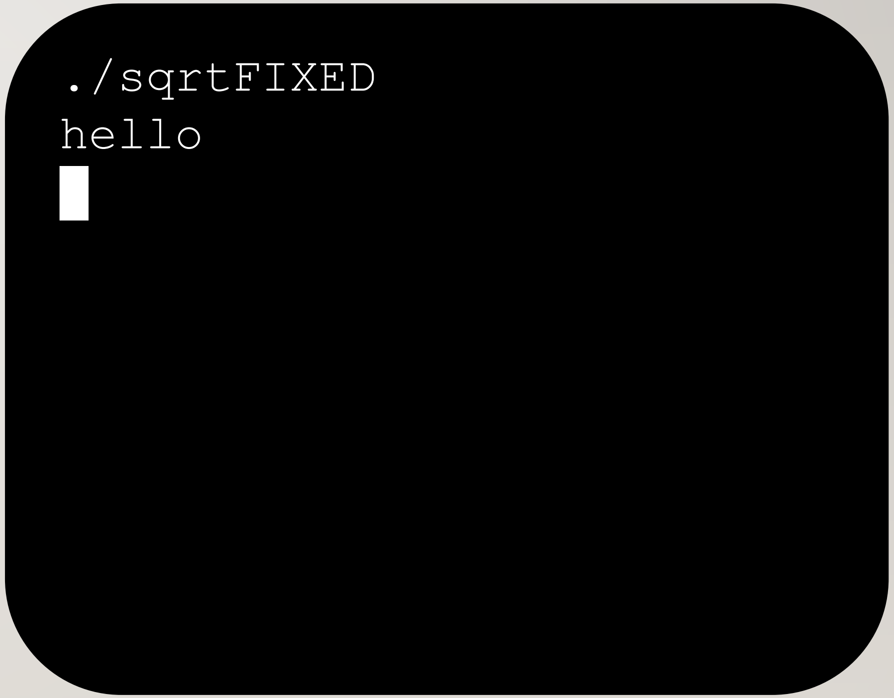


```
./sqrtFIXED
█
```

GO TO

```
GO TO para1.
stop run.

para1.
    display "hello".

para2.
    display "world".
```

GO TO means **GO TO AND NEVER COME BACK.**



```
./sqrtFIXED
hello
█
```

GO TO

```
GO TO para1.
stop run.

para1.
    display "hello".

para2.
    display "world".
```

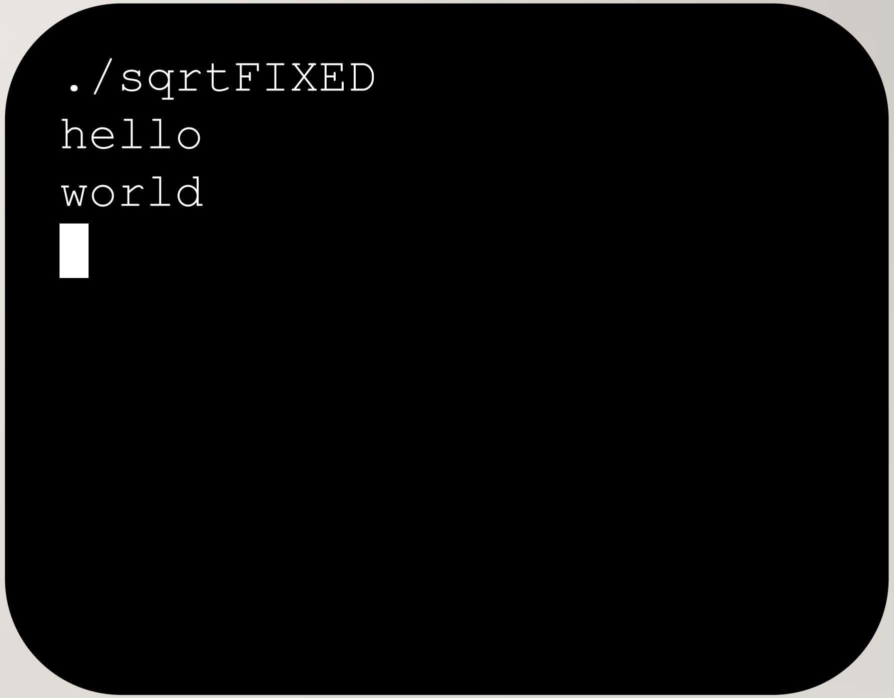
It keeps going forward after. Like if you physically moved the program counter

```
./sqrtFIXED
hello
█
```

GO TO

```
GO TO para1.
stop run.

para1.
    display "hello".

para2.
    display "world".
```

It keeps going forward after. Like if you physically moved the program counter



```
./sqrtFIXED
hello
world
█
```

GO TO

```
GO TO para1.
stop run.

para1.
    display “hello”.

para2.
    display “world”.
```

Program terminates.

```
./sqrtFIXED
hello
world
masterOf@COBOL:~ █
```

PERFORM VS. GOTO

- **PERFORM** is like a function call
  - Alice: "**PERFORM** the dance for me." Bob: "OK, I just did it; now what?
  - When you (Bob) are **performing** something, you are doing it for another entity (Alice). This entity knows you are going to return to them and ask for the next instruction.

- **GO TO** is an unconditional jump
  - Alice: "**GO TO** work" *Bob goes to work and asks Charlie for work*
  - When you (Bob) are told to **go to** somewhere else by another entity (Alice), this entity can no longer issue instructions to you after, because they **gave up control of you**. They can do whatever!

PERFORM IS LIKE A YOYO – ALWAYS COMES BACK

perform PARA1.

PERFORM IS LIKE A YOYO – ALWAYS COMES BACK

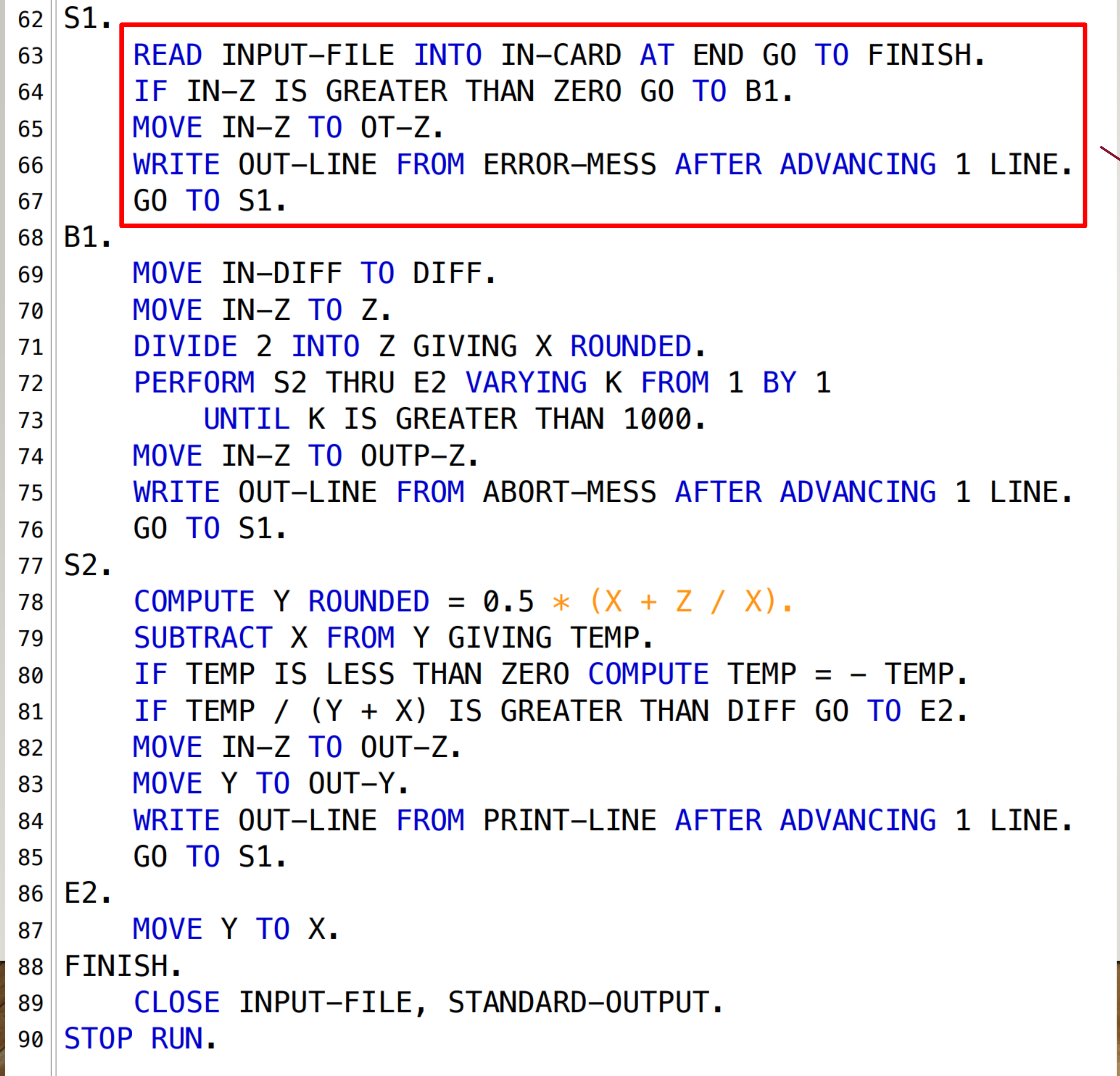
```
perform PARA1.


perform PARA2.
```

PERFORM IS LIKE A YOYO – ALWAYS COMES BACK

```
perform PARA1.

perform PARA2.

perform PARA3.
```

GO TO IS LIKE…CUTTING TIES WITH SOMEONE

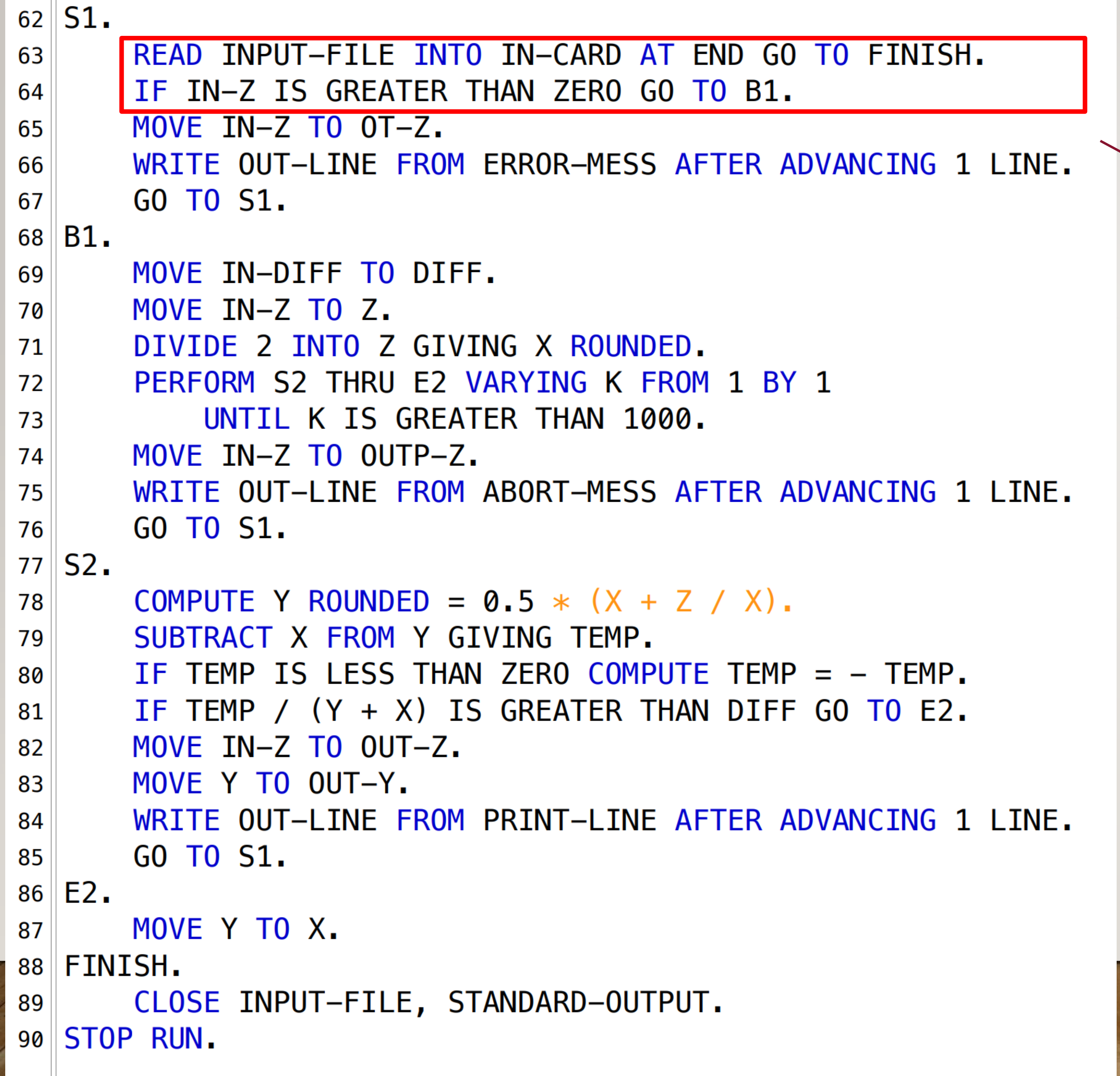
go to PARA1.



```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO GO TO B1.
    MOVE IN-Z TO OT-Z.
    WRITE OUT-LINE FROM ERROR-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
B1.
    MOVE IN-DIFF TO DIFF.
    MOVE IN-Z TO Z.
    DIVIDE 2 INTO Z GIVING X ROUNDED.
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF TEMP / (Y + X) IS GREATER THAN DIFF GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

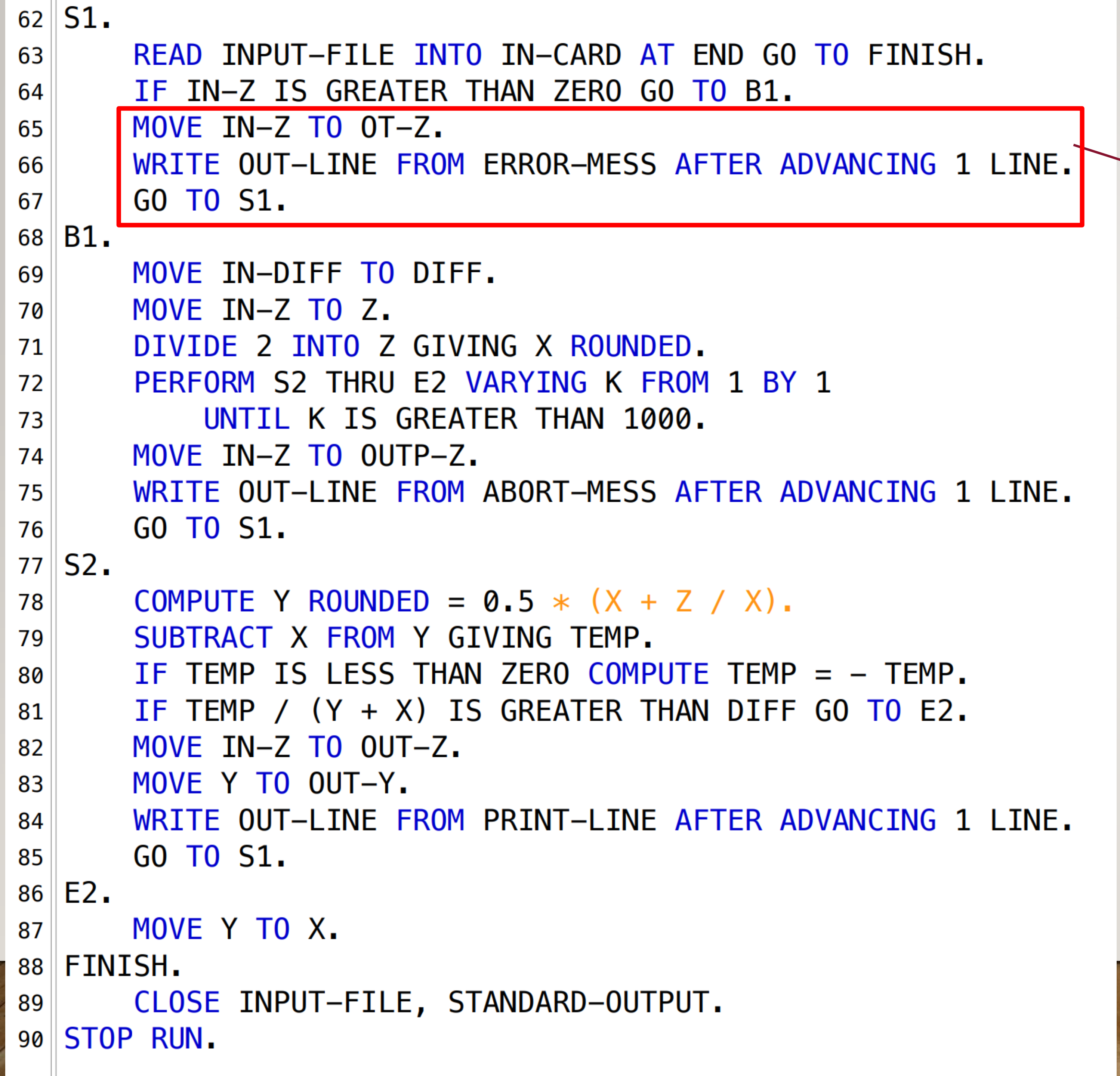
I am going to abstract away and black-box some useless parts of the program throughout this PowerPoint. First…

The **S1** paragraph can be bastardized as "**READ_A_LINE**": Here's why.



```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO GO TO B1.
    MOVE IN-Z TO OT-Z.
    WRITE OUT-LINE FROM ERROR-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
B1.
    MOVE IN-DIFF TO DIFF.
    MOVE IN-Z TO Z.
    DIVIDE 2 INTO Z GIVING X ROUNDED.
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF TEMP / (Y + X) IS GREATER THAN DIFF GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Basically it reads in a line and if the input number (**IN-Z**) is positive (aka can be square rooted), we **GO TO B1** which is the calculation paragraph.



```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO GO TO B1.
    MOVE IN-Z TO OT-Z.
    WRITE OUT-LINE FROM ERROR-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
B1.
    MOVE IN-DIFF TO DIFF.
    MOVE IN-Z TO Z.
    DIVIDE 2 INTO Z GIVING X ROUNDED.
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF TEMP / (Y + X) IS GREATER THAN DIFF GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Otherwise the program flow just falls through, printing the **ERROR-MESSAGE** and then requesting another line (**GO TO S1**).

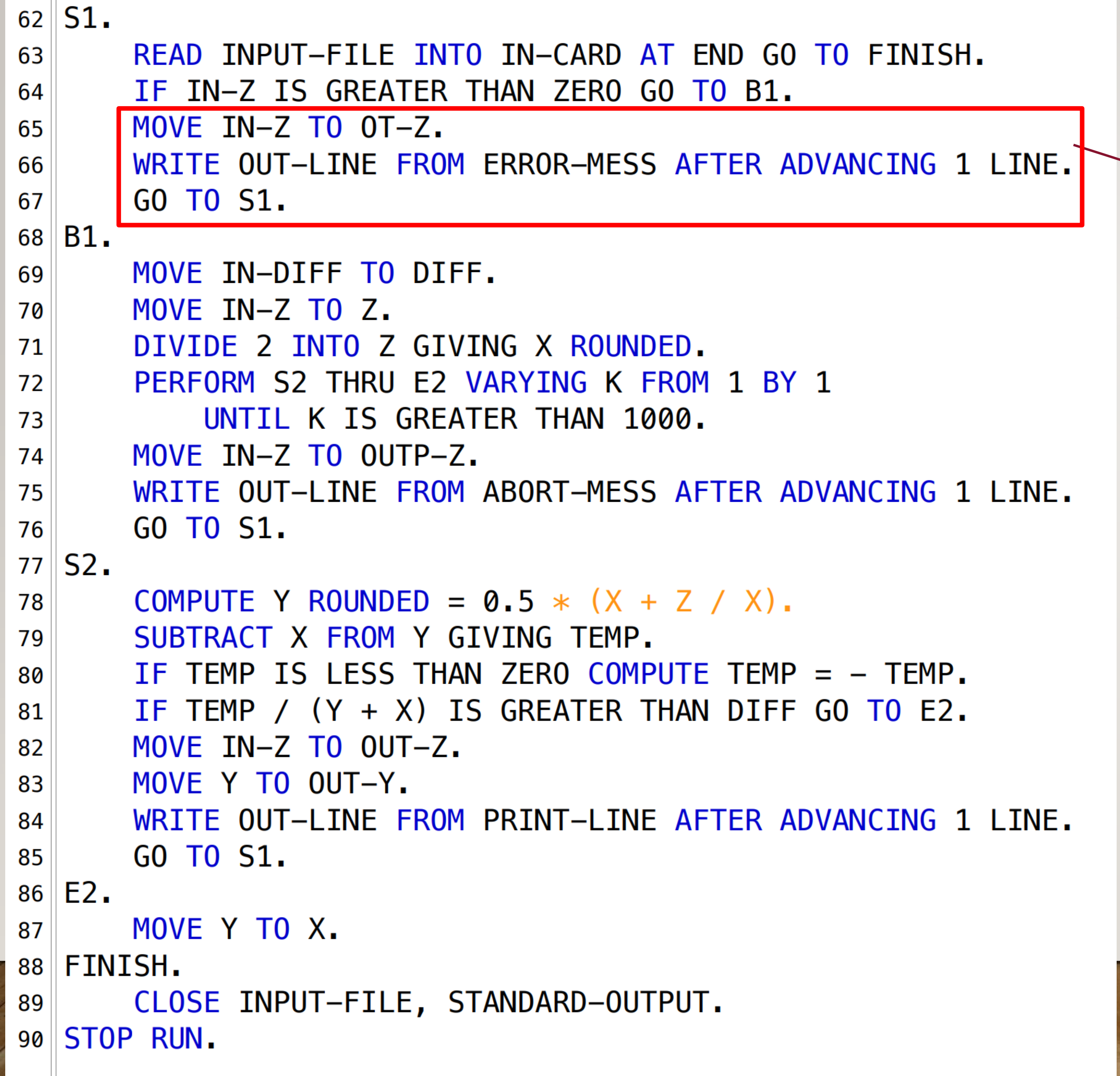
AH YES, FALLTHROUGH

- Whereas most programs today would use two paths in a fork when enforcing control flow like this:

```
while (true) {
    if (cond)
        printf(“Success\n”);
    else
        printf(“Error\n”);
}
```

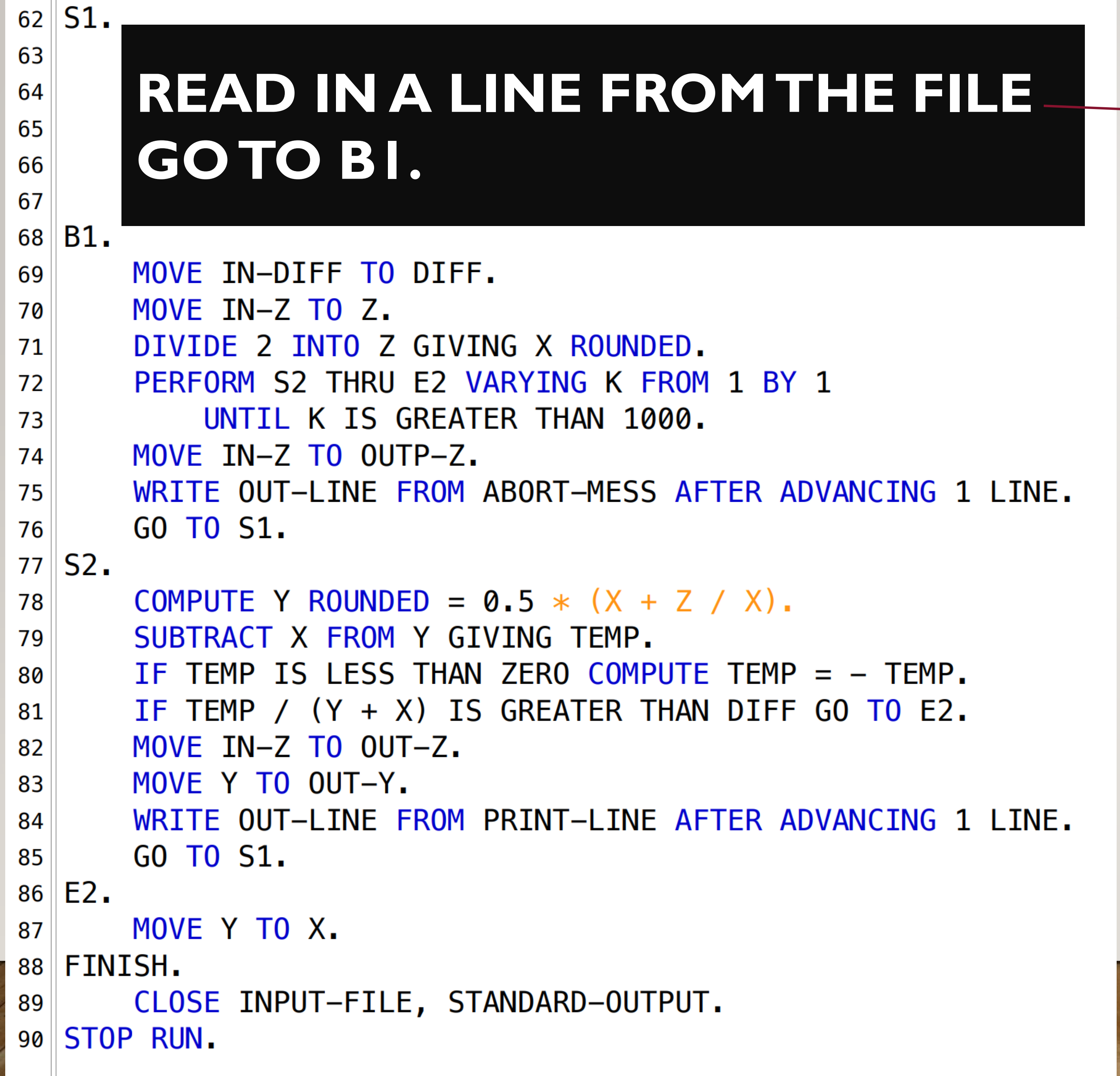
- …logic back then would only have one path, and if you didn’t go to that path you would **fall through** to an alternate one

```
S1:
    if (cond) goto SUCCESS;
    printf(“Error\n”);
    goto S1;

SUCCESS:
    printf(“Success\n”);
    goto S1;
```



```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO GO TO B1.
    MOVE IN-Z TO OT-Z.
    WRITE OUT-LINE FROM ERROR-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
B1.
    MOVE IN-DIFF TO DIFF.
    MOVE IN-Z TO Z.
    DIVIDE 2 INTO Z GIVING X ROUNDED.
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF TEMP / (Y + X) IS GREATER THAN DIFF GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

As you can see here, **B1** is the success path, and if you don't reach it, you just **fall through** to the **"INVALID INPUT"** path.



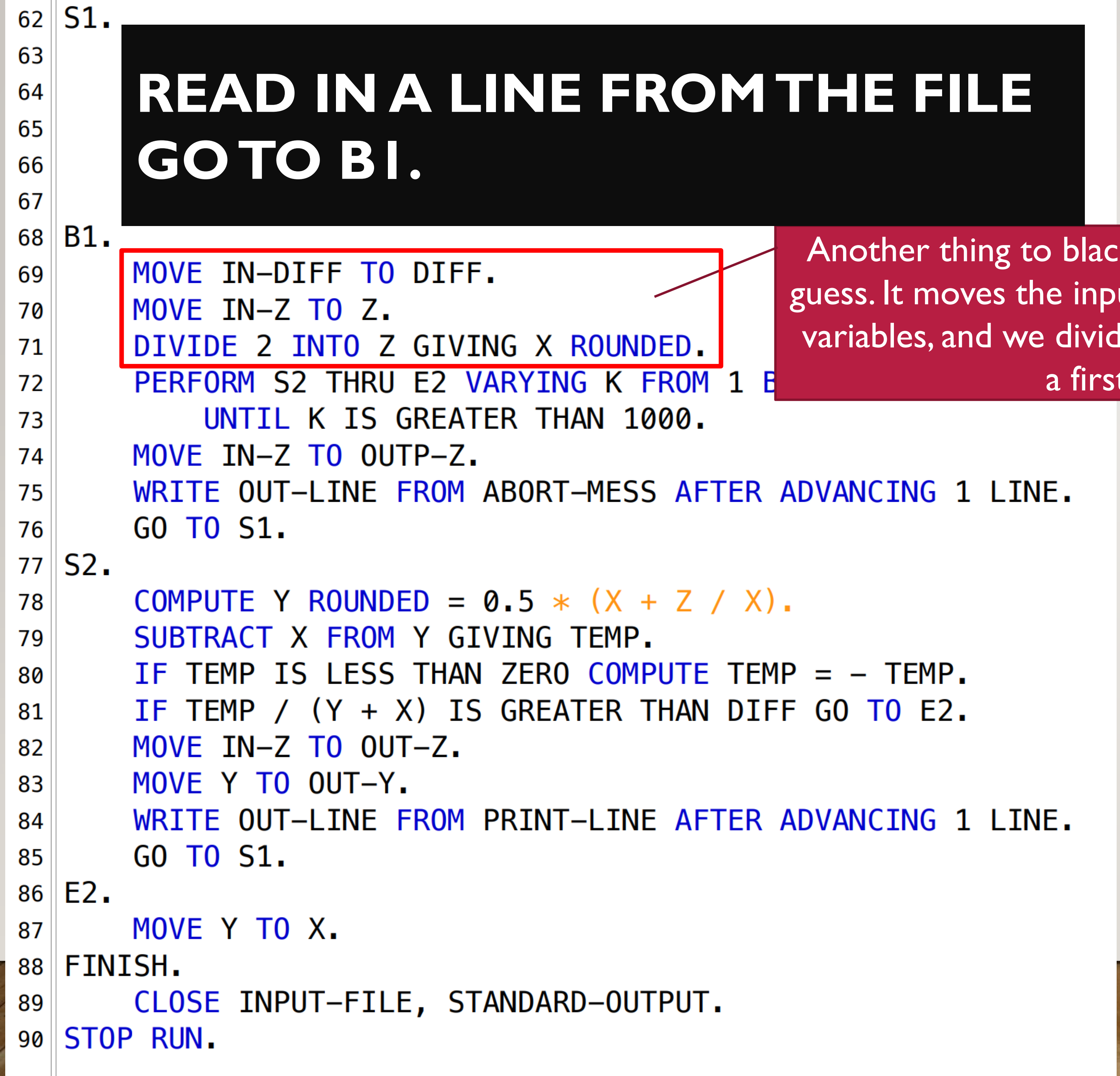
```
S1.
```

READ IN A LINE FROM THE FILE GO TO B1.

```
B1.
    MOVE IN-DIFF TO DIFF.
    MOVE IN-Z TO Z.
    DIVIDE 2 INTO Z GIVING X ROUNDED.
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF TEMP / (Y + X) IS GREATER THAN DIFF GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

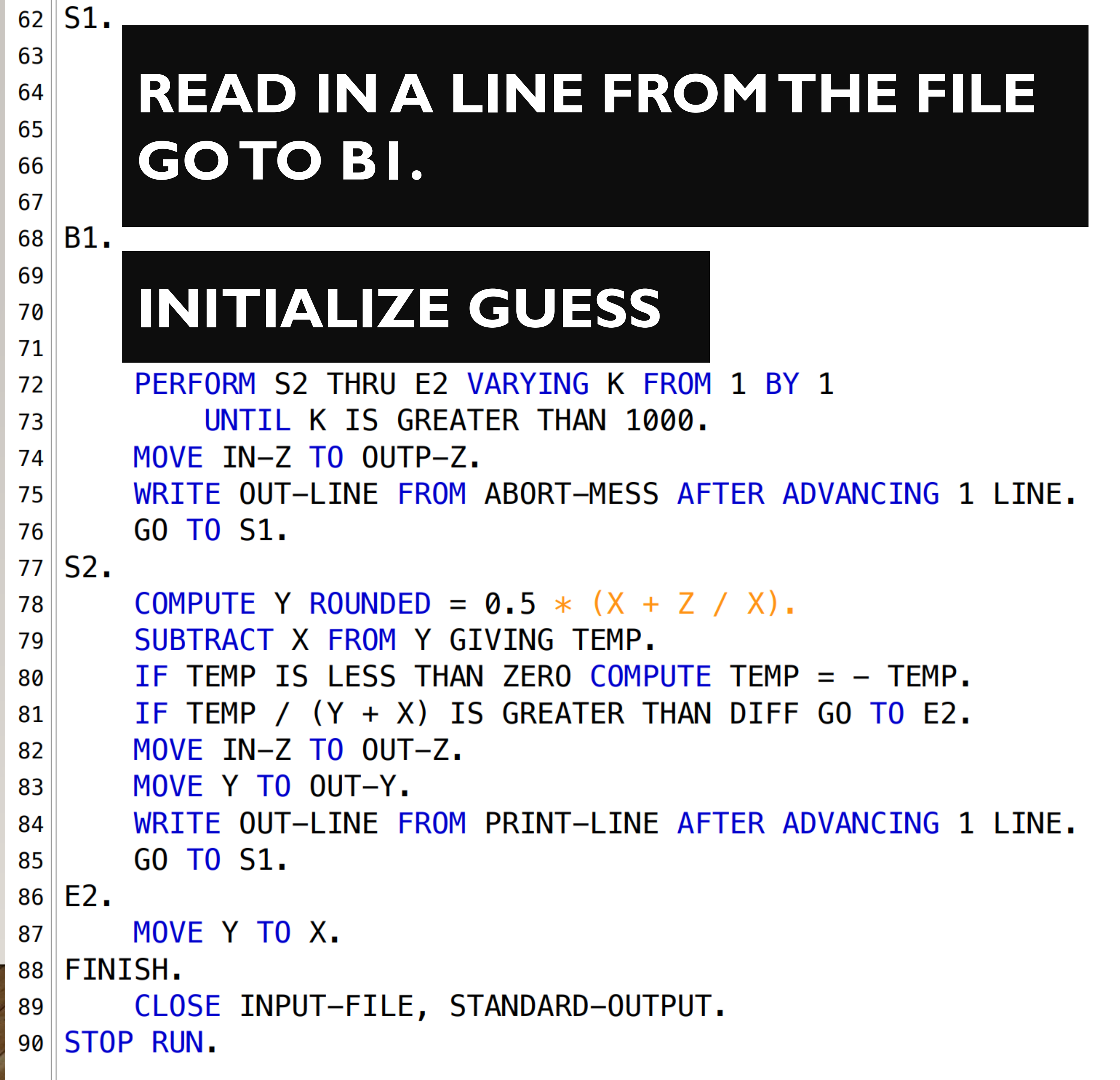
Let’s blackbox and simplify **S1** then, assuming that the error doesn’t occur. We don’t need that to understand later parts.

READ IN A LINE FROM THE FILE
GO TO B1.



```
S1.
B1.
    MOVE IN-DIFF TO DIFF.
    MOVE IN-Z TO Z.
    DIVIDE 2 INTO Z GIVING X ROUNDED.
    PERFORM S2 THRU E2 VARYING K FROM 1 B
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF TEMP / (Y + X) IS GREATER THAN DIFF GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Another thing to blackbox: this initializes the guess. It moves the input variables to our work variables, and we divide our number in half as a first guess



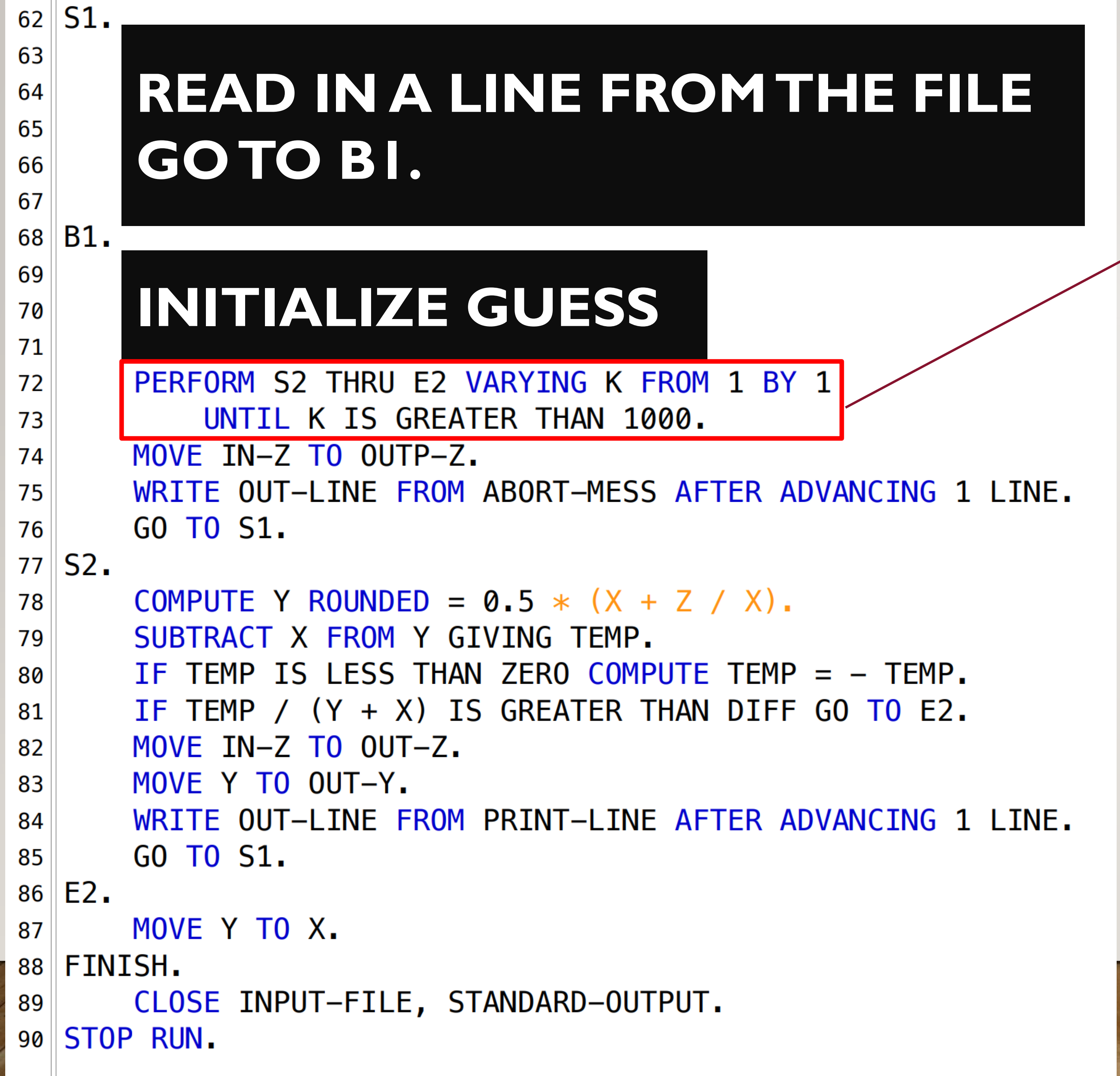
```
S1.
```

READ IN A LINE FROM THE FILE GO TO B1.

```
B1.
```

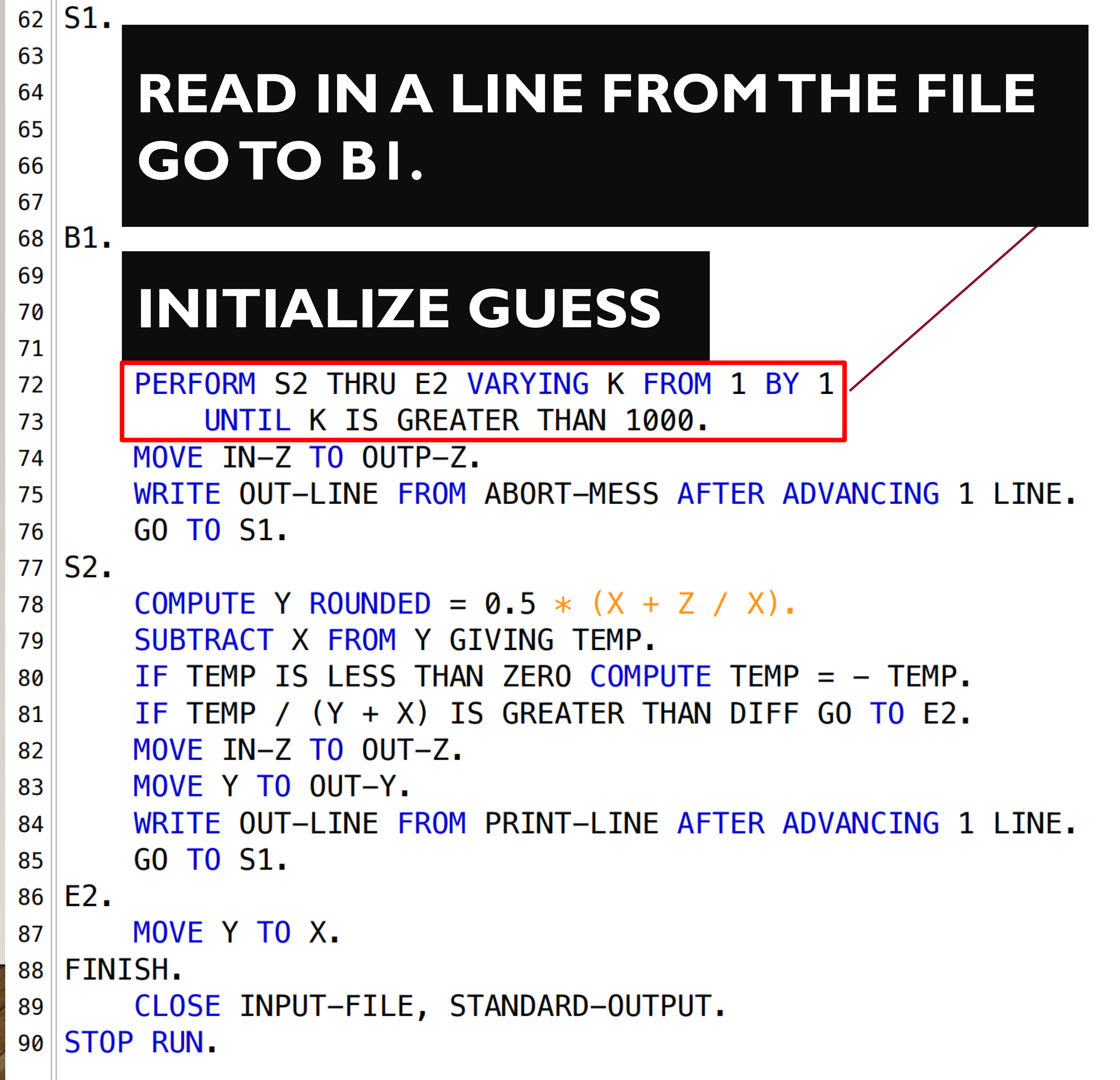
INITIALIZE GUESS

```
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF TEMP / (Y + X) IS GREATER THAN DIFF GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```



```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF TEMP / (Y + X) IS GREATER THAN DIFF GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

28 slides later, the main character of this PowerPoint comes out.



```
S1.
```

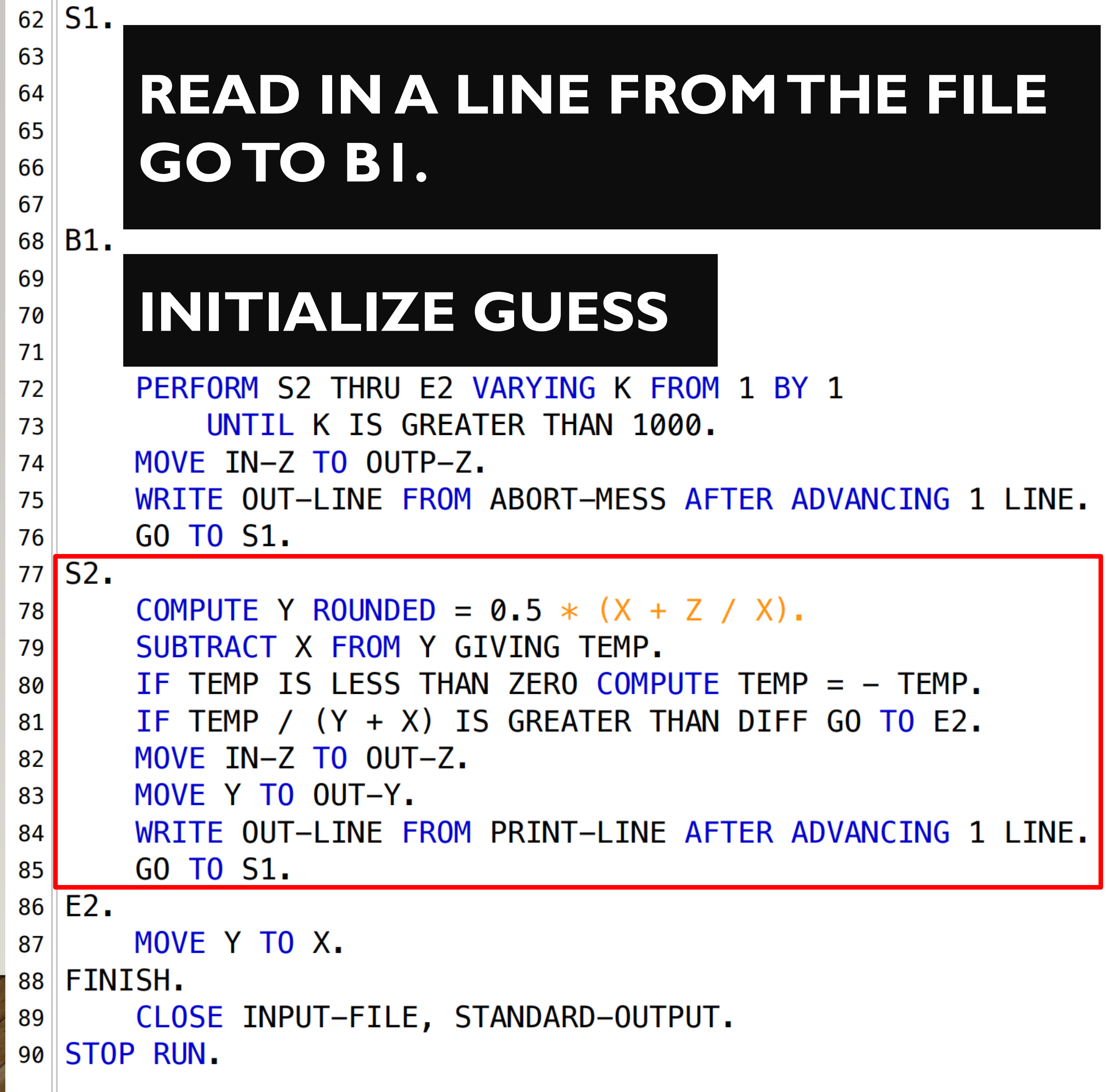
READ IN A LINE FROM THE FILE
GO TO B1.

```
B1.
```

INITIALIZE GUESS

```
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF TEMP / (Y + X) IS GREATER THAN DIFF GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Let’s take things step by step.



```
S1.
```

READ IN A LINE FROM THE FILE GO TO B1.

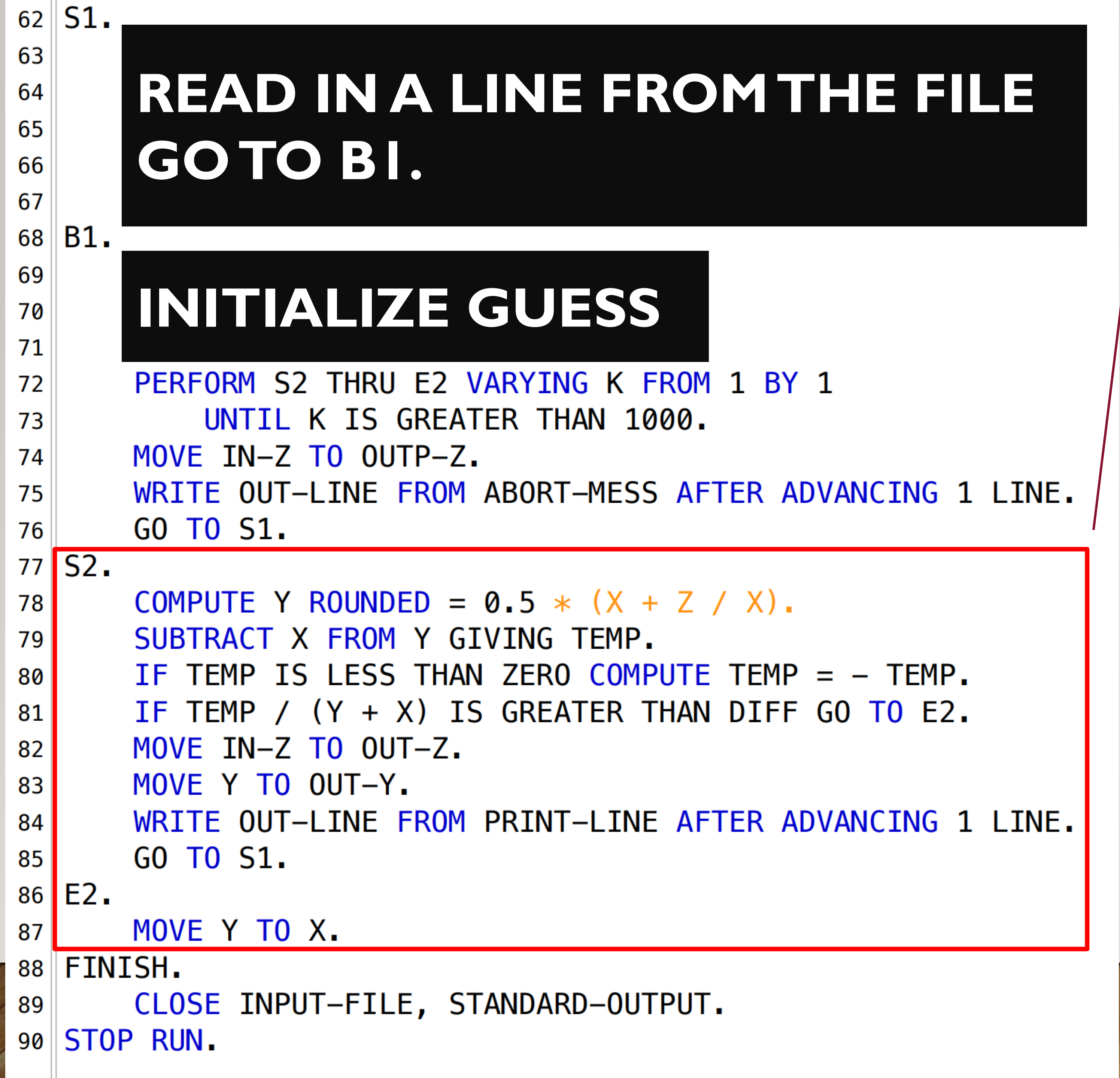
```
B1.
```

INITIALIZE GUESS

```
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF TEMP / (Y + X) IS GREATER THAN DIFF GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Let's take things step by step.

PERFORM S2 will just do the **S2** paragraph and then return to the calling scope. This will do only one iteration for every line of the file. So if you made a mistake re-engineering and you call only **S2**, you might get **2.25** instead of **2.23** for sqrt(5)



```
S1.
```

READ IN A LINE FROM THE FILE GO TO B1.

```
B1.
```

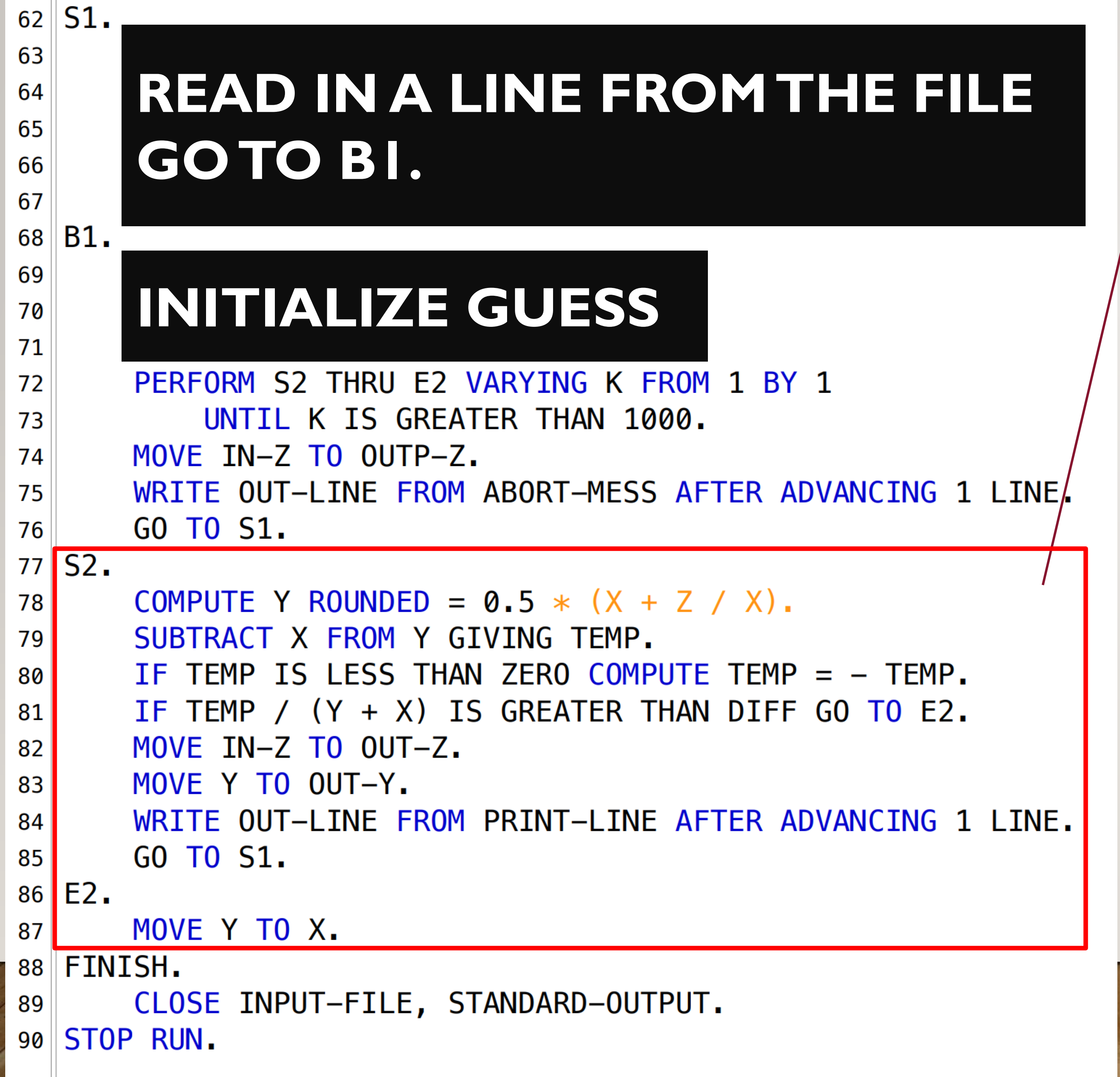
INITIALIZE GUESS

```
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF TEMP / (Y + X) IS GREATER THAN DIFF GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Let's take things step by step.

PERFORM S2 will just do the **S2** paragraph and then return to the calling scope. This will do only one iteration for every line of the file. So if you made a mistake re-engineering and you call only **S2**, you might get **2.25** instead of **2.23** for sqrt(5)

PERFORM S2 THRU E2 will do both **S2** and **E2**. When **S2** finishes and returns to calling scope, the **PERFORM** loop will then **PERFORM E2** (detailed explanation in a second).



```
S1.
```

READ IN A LINE FROM THE FILE GO TO B1.

```
B1.
```

INITIALIZE GUESS

```
     PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
         UNTIL K IS GREATER THAN 1000.
     MOVE IN-Z TO OUTP-Z.
     WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
     GO TO S1.
S2.
     COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
     SUBTRACT X FROM Y GIVING TEMP.
     IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
     IF TEMP / (Y + X) IS GREATER THAN DIFF GO TO E2.
     MOVE IN-Z TO OUT-Z.
     MOVE Y TO OUT-Y.
     WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
     GO TO S1.
E2.
     MOVE Y TO X.
FINISH.
     CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Let's take things step by step.

PERFORM S2 will just do the **S2** paragraph and then return to the calling scope. This will do only one iteration for every line of the file. So if you made a mistake re-engineering and you call only **S2**, you might get **2.25** instead of **2.23** for sqrt(5)

PERFORM S2 THRU E2 will do both **S2** and **E2**. When **S2** finishes and returns to calling scope, the **PERFORM** loop will then **PERFORM E2** (detailed explanation in a second).

PERFORM S2 THRU E2 VARYING K FROM 1 BY 1 UNTIL K IS GREATER THAN 1000 is the same as above, except it will **continually** call **S2** and **E2** while keeping count using variable **K**, stopping when it is at 1000:

```
for (int K = 1; K <= 1000; K++) {
        performS2();
        performE2();
}
```

WELL TECHNICALLY YOU’RE RIGHT BUT…

- In a **PERFORM THRU** statement, you specify a START and an END. Then you do everything in between.

```
PERFORM  FIRST  THRU  LAST  1000  TIMES.
STOP RUN.

FIRST.
   DISPLAY “WHY”.

SECOND.
   DISPLAY “HELLO”.

THIRD.
   DISPLAY “THERE”.

LAST.
   DISPLAY “WORLD”.
```

WELL TECHNICALLY YOU'RE RIGHT BUT…

- In a **PERFORM THRU** statement, you specify a START and an END. Then you do everything in between.

```
PERFORM FIRST THRU LAST 1000 TIMES.
STOP RUN.

FIRST.
   DISPLAY “WHY”.

SECOND.
   DISPLAY “HELLO”.

THIRD.
   DISPLAY “THERE”.

LAST.
   DISPLAY “WORLD”.
```

WELL TECHNICALLY YOU’RE RIGHT BUT…

- In a **PERFORM THRU** statement, you specify a START and an END. Then you do everything in between.

```
PERFORM FIRST THRU LAST 1000 TIMES.
STOP RUN.

FIRST.
   DISPLAY “WHY”.

SECOND.
   DISPLAY “HELLO”.

THIRD.
   DISPLAY “THERE”.

LAST.
   DISPLAY “WORLD”.
```

WELL TECHNICALLY YOU'RE RIGHT BUT…

- In a **PERFORM THRU** statement, you specify a START and an END. Then you do everything in between.

```
PERFORM FIRST THRU LAST 1000 TIMES.
STOP RUN.
```

```
FIRST.
   DISPLAY "WHY".

SECOND.
   DISPLAY "HELLO".

THIRD.
   DISPLAY "THERE".

LAST.
   DISPLAY "WORLD".
```

IS THE

SAME AS

WELL TECHNICALLY YOU'RE RIGHT BUT…

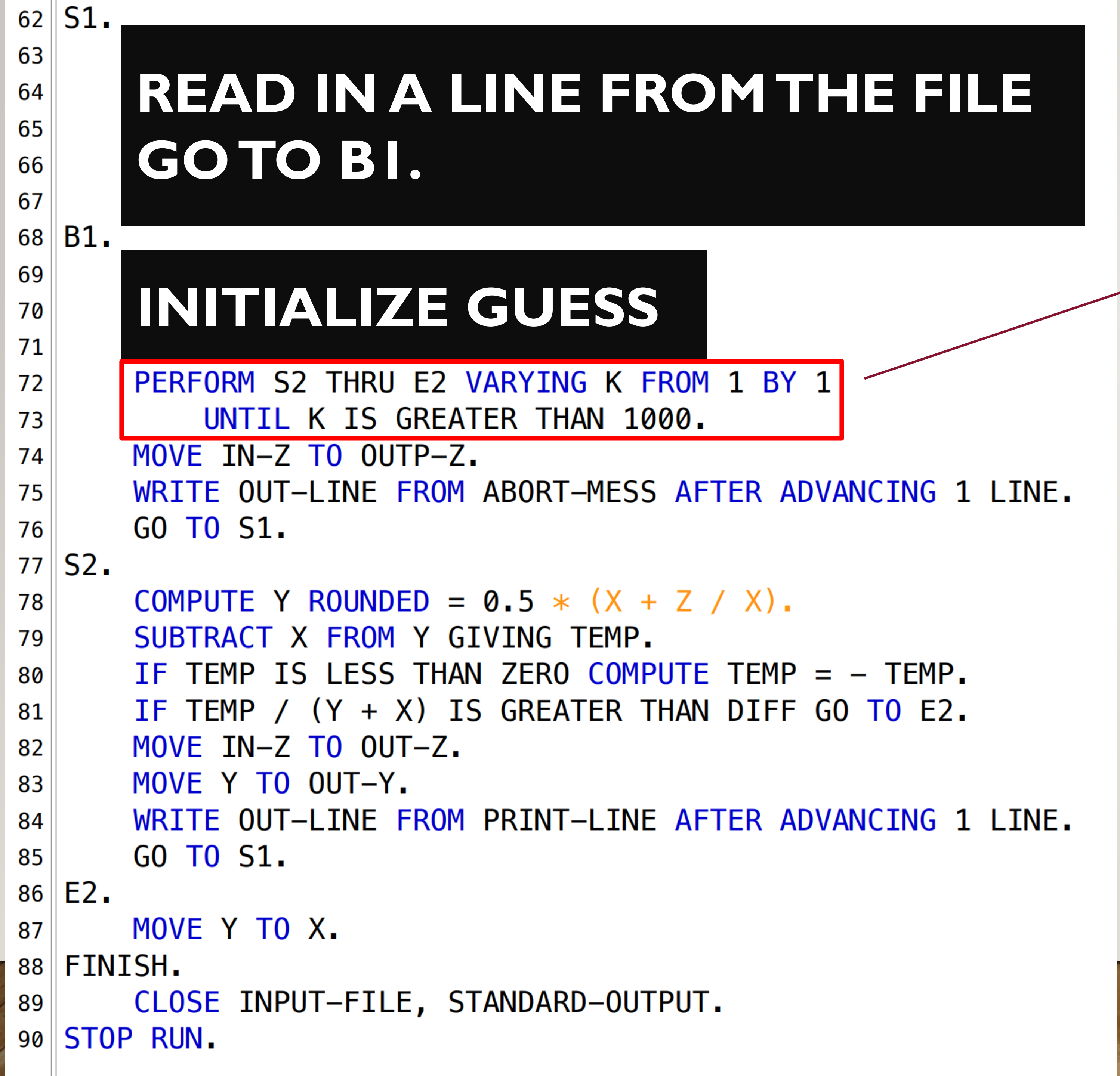
- In a **PERFORM THRU** statement, you specify a START and an END. Then you do everything in between.

```
PERFORM FIRST THRU LAST 1000 TIMES.
STOP RUN.
```

```
FIRST.
   DISPLAY "WHY".

SECOND.
   DISPLAY "HELLO".

THIRD.
   DISPLAY "THERE".

LAST.
   DISPLAY "WORLD".
```

IS THE SAME AS

```
PERFORM COMBINED 1000 TIMES.
STOP RUN.

COMBINED.
   DISPLAY "WHY".
   DISPLAY "HELLO".
   DISPLAY "THERE".
   DISPLAY "WORLD".
```



```
S1.
```

READ IN A LINE FROM THE FILE GO TO B1.

```
B1.
```

INITIALIZE GUESS

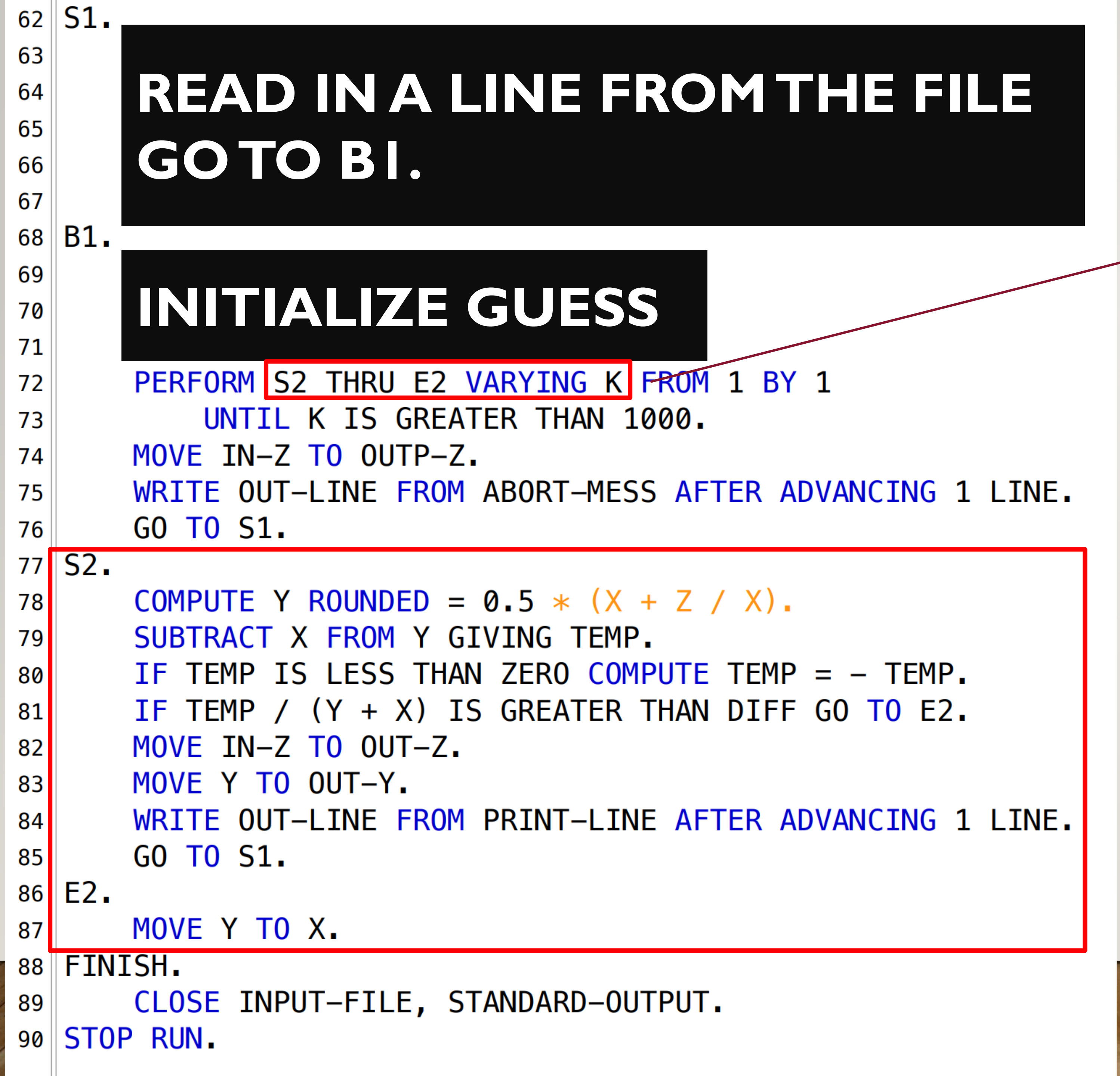
```
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF TEMP / (Y + X) IS GREATER THAN DIFF GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

This **for** loop seems to sum it up:

```
for (int K = 1; K <= 1000; K++) {
        performS2();
        performE2();
}
```

Except… that above statement assumes **performS2()** ends at the end of **performS2()**.

In reality, if the **performS2()** function has a **GO TO** statement that spits you out somewhere else in the program…this **for** loop is powerless to know that, because **GO TO gives up control permanently**. This is like using **break;**



```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.

B1.
    INITIALIZE GUESS

    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF TEMP / (Y + X) IS GREATER THAN DIFF GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Recall that this **PERFORM** loop glues multiple paragraphs together into a new scope. So in a sense we are gluing together **S2** and **E2**, which results in a massive scope consisting of both paragraphs.

This means you can **GO TO** anywhere between the glued-together **S2-E2** paragraph and you'd still be within the **PERFORM** statement's grasp (you wouldn't destroy its scope as stated earlier).

However…not all **GO TO**s play by the rules like this (**GO TO** last slide)

INTERMISSION

```
S1.
```

READ IN A LINE FROM THE FILE GO TO B1.

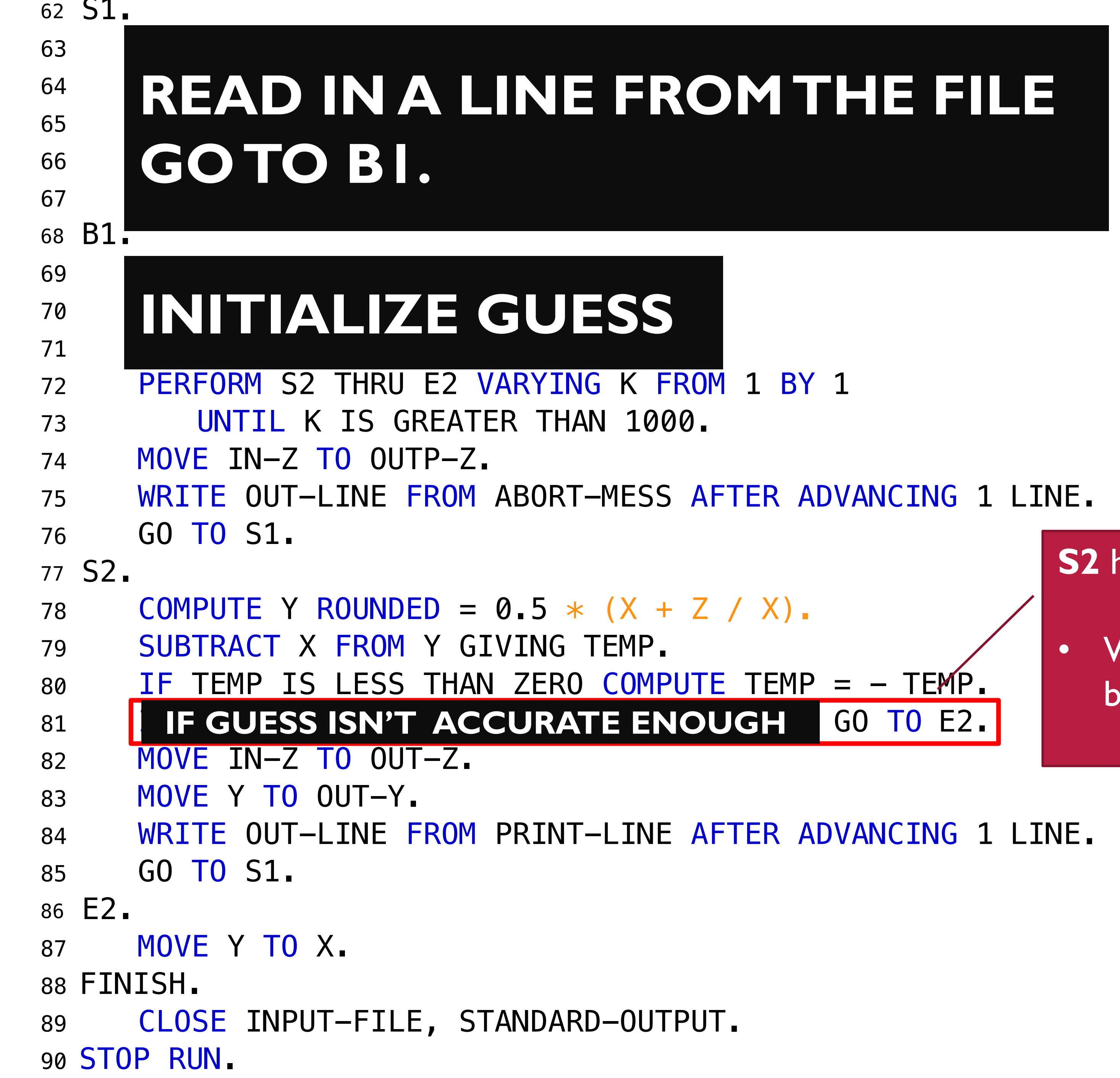
```
B1.
```

INITIALIZE GUESS

```
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
       UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF TEMP / (Y + X) IS GREATER THAN DIFF GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

S2 has two ways of ending:

- When it decides to jump to **E2** because it wants more accuracy



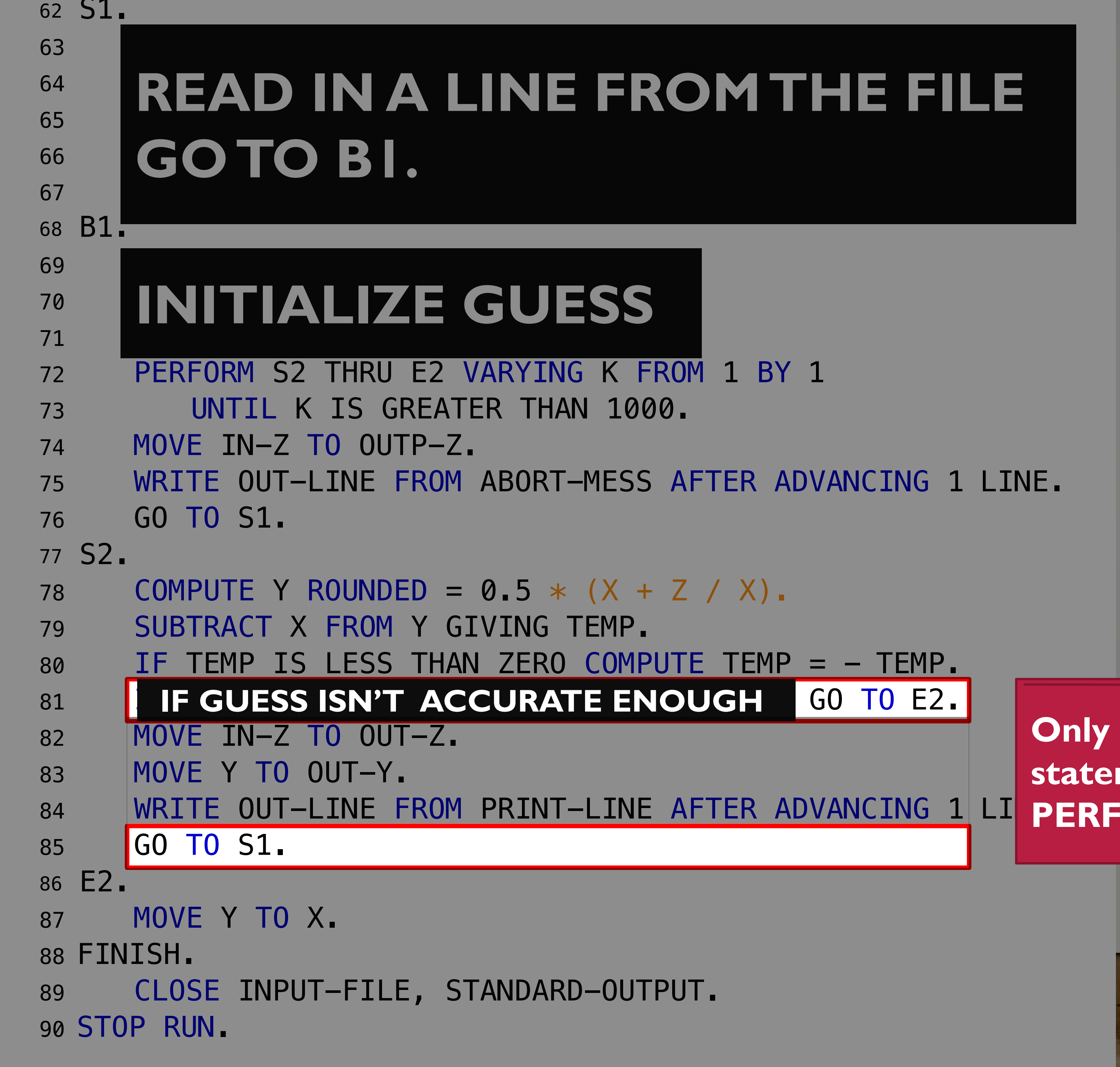
```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

S2 has two ways of ending:

- When it decides to jump to **E2** because it wants more accuracy

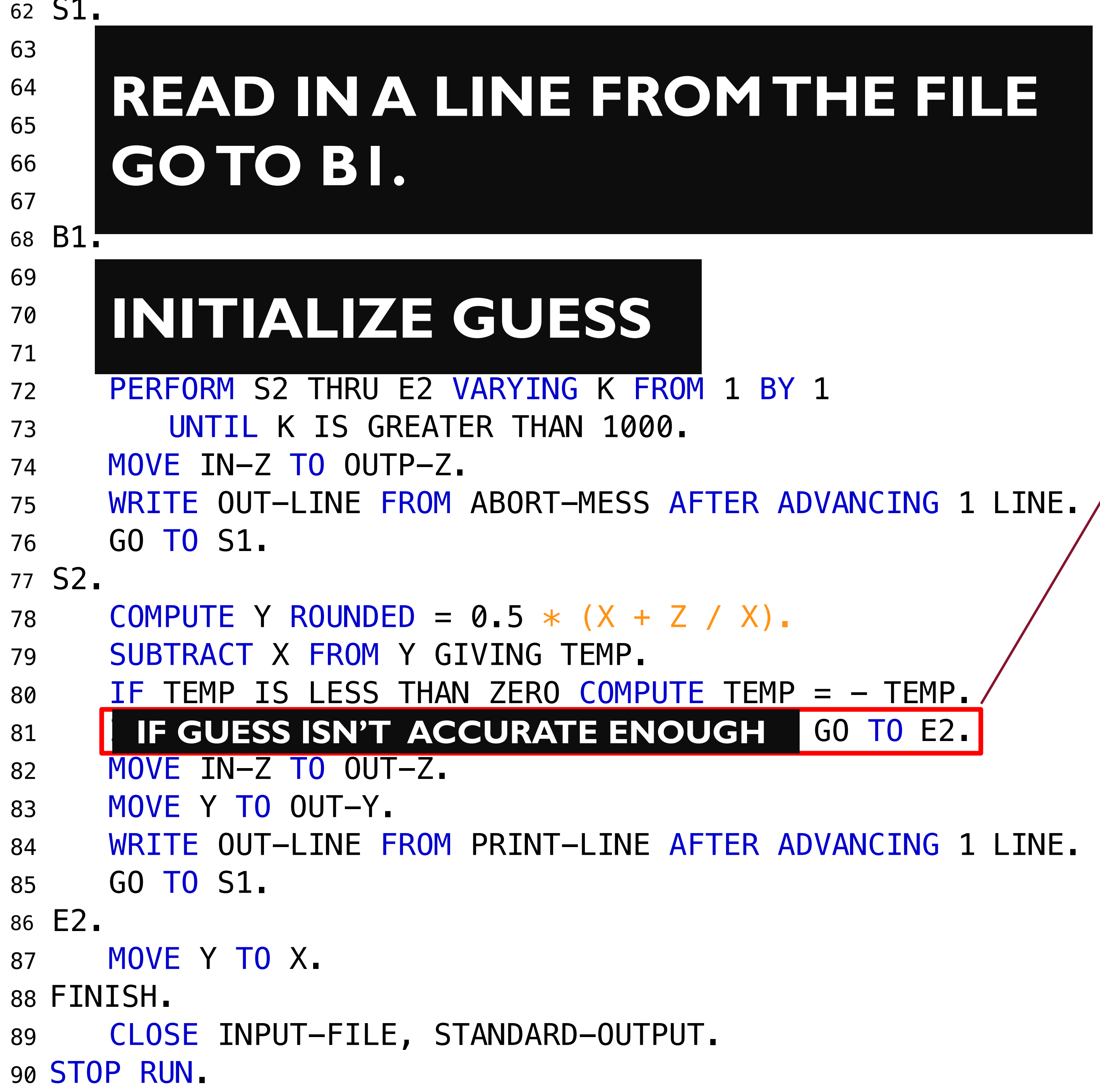
```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
       UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
     IF GUESS ISN'T  ACCURATE ENOUGH  GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LI
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

- Or when the accuracy check doesn't fail (which would mean **GO TO E2**), resulting in fallthrough: it prints the output line and then reads in a new line; which means **GO TO S1**.



```
S1.

    READ IN A LINE FROM THE FILE
    GO TO B1.

B1.

    INITIALIZE GUESS

    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LI
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Only one of these two GO TO statements respects the PERFORM LOOP scope.



```
S1.

    READ IN A LINE FROM THE FILE
    GO TO B1.

B1.

    INITIALIZE GUESS

    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

It's this one.

Remember that **S2** and **E2** are effectively **glued** for the remainder of the **PERFORM** call.

This means **GO TO E2** still puts you in the **PERFORM** statement's territory, which is anything in **S2/E2**

```
S1.

    READ IN A LINE FROM THE FILE
    GO TO B1.

B1.

    INITIALIZE GUESS

    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

It's the first one.

Remember that **S2** and **E2** are effectively **glued** for the remainder of the **PERFORM** call!!!

This means **GO TO E2** still puts you in the **PERFORM** statement's territory, which is anything in **S2/E2**

This is our turf.

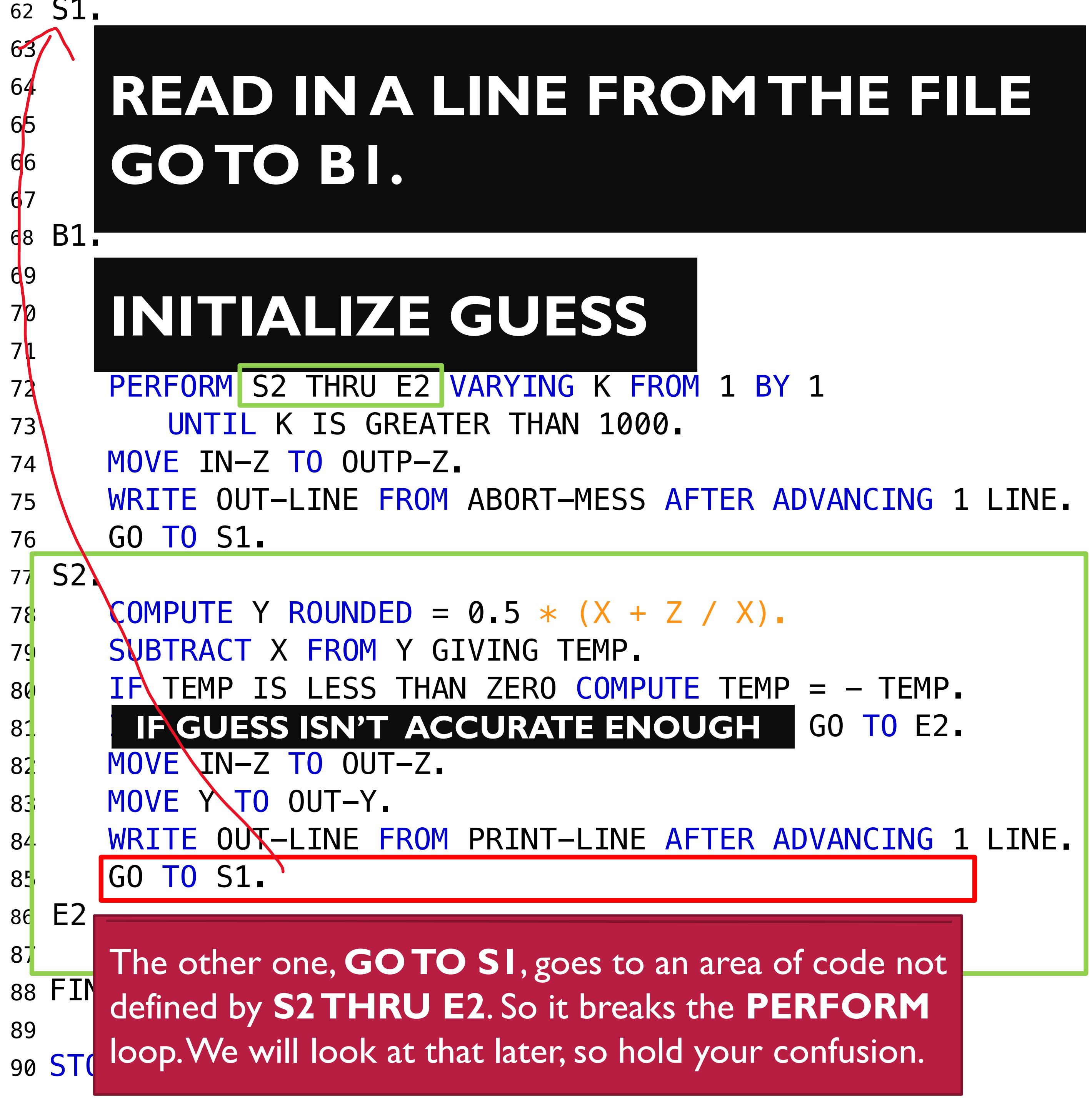
```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
       UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Still our turf!!!

It's the first one.

Remember that **S2** and **E2** are effectively **glued** for the remainder of the **PERFORM** call!!!

This means **GO TO E2** still puts you in the **PERFORM** statement's territory, which is anything in **S2/E2**

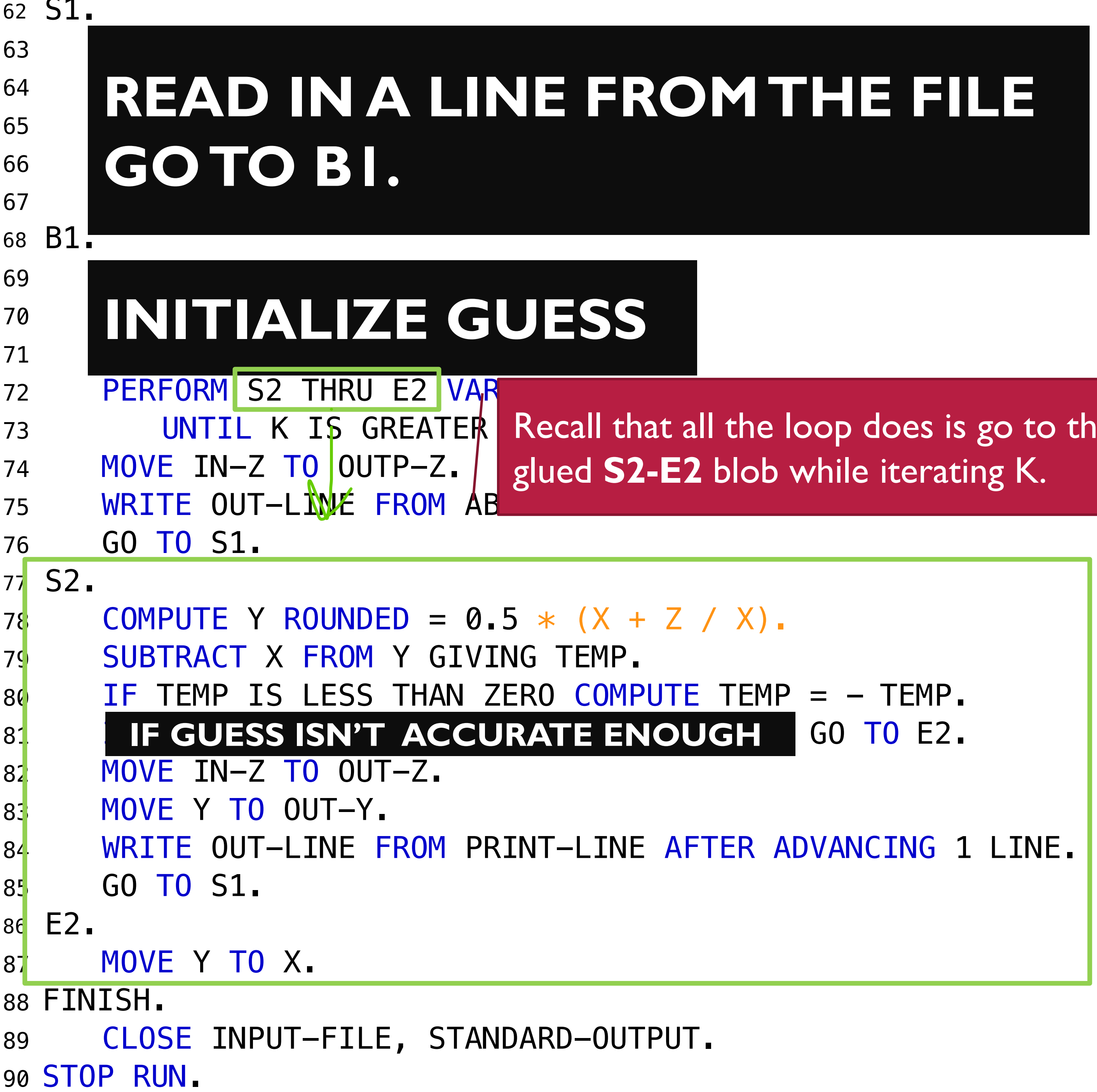


```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
       UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2
FIN
STO
```

The other one, **GO TO S1**, goes to an area of code not defined by **S2 THRU E2**. So it breaks the **PERFORM** loop. We will look at that later, so hold your confusion.

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VAR
       UNTIL K IS GREATER
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM AB
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
     IF GUESS ISN'T  ACCURATE ENOUGH  GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Recall that all the loop does is go to the glued **S2-E2** blob while iterating K.

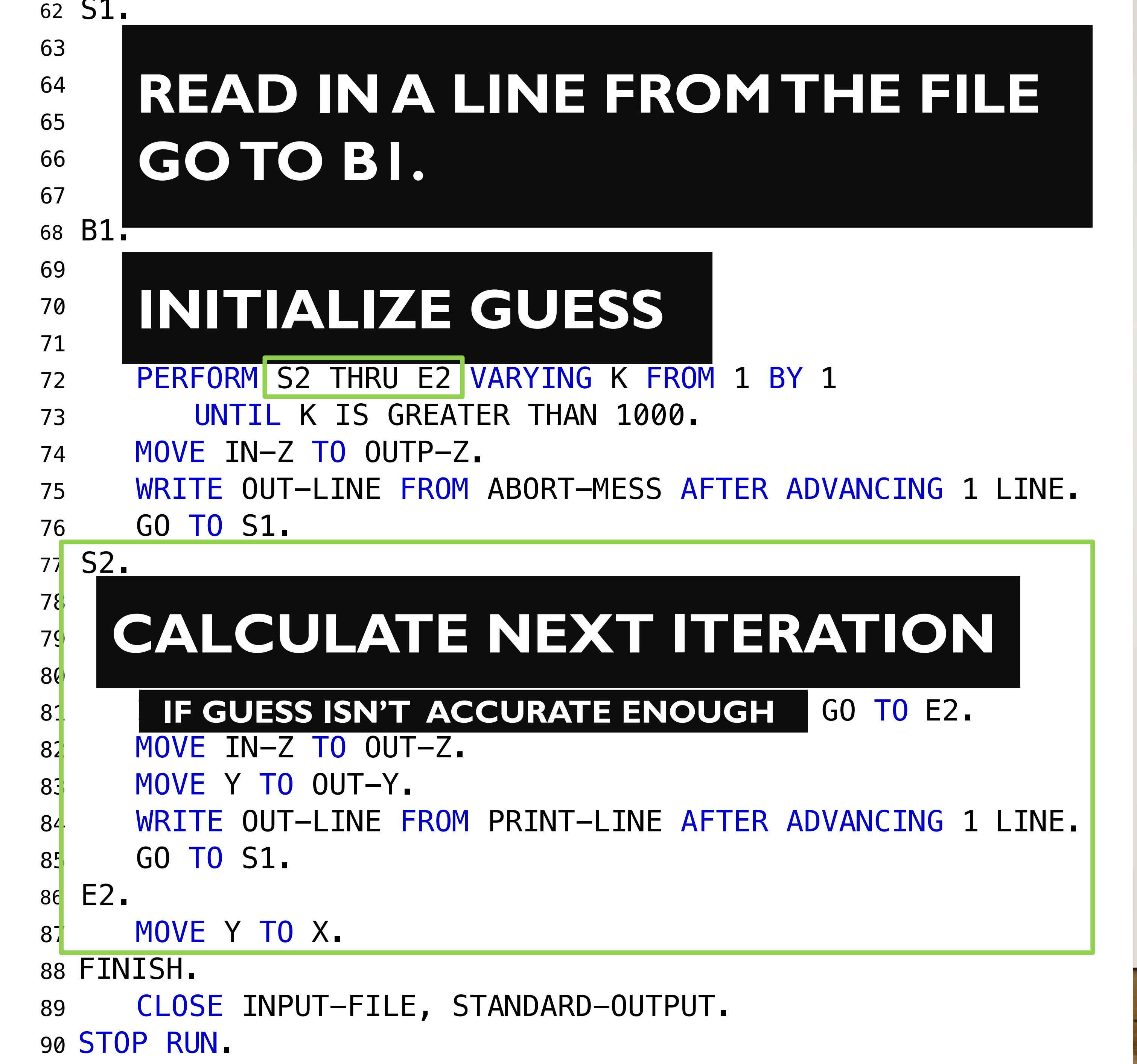


```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VAR
       UNTIL K IS GREATER
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM AB
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
     IF GUESS ISN'T  ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Recall that all the loop does is go to the glued **S2-E2** blob while iterating K.

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

These three lines just calculate another iteration of Babylon. Let me abstract it out for you owo



```
S1.
```

READ IN A LINE FROM THE FILE
GO TO B1.

```
B1.
```

INITIALIZE GUESS

```
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
       UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
```

CALCULATE NEXT ITERATION

```
    IF GUESS ISN’T  ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO
    WRITE OUT-LI                                     ANCING 1 LINE.
    GO TO S1.
S2.
   CALCULATE NEXT ITERATION
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Here is usual program flow.

```
S1.
   READ IN A LINE FROM THE FILE
   GO TO B1.
B1.
   INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO
    WRITE OUT-LI                               ANCING 1 LINE.
    GO TO S1.
S2.
   CALCULATE NEXT ITERATION
    IF GUESS ISN'T  ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

The for loop (**PERFORM**) executes the glued **S2-E2** block

```
S1.

    READ IN A LINE FROM THE FILE
    GO TO B1.

B1.

    INITIALIZE GUESS

    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO
    WRITE OUT-LI                                   ANCING 1 LINE.
    GO TO S1.
S2.

   CALCULATE NEXT ITERATION

     IF GUESS ISN'T  ACCURATE ENOUGH   GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

The for loop (**PERFORM**) executes the glued **S2-E2** block

```
S1.
```

READ IN A LINE FROM THE FILE GO TO B1.

```
B1.
```

INITIALIZE GUESS

```
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO
    WRITE OUT-LI                               ANCING 1 LINE.
    GO TO S1.
```

Calculating next iteration…

```
S2.
```

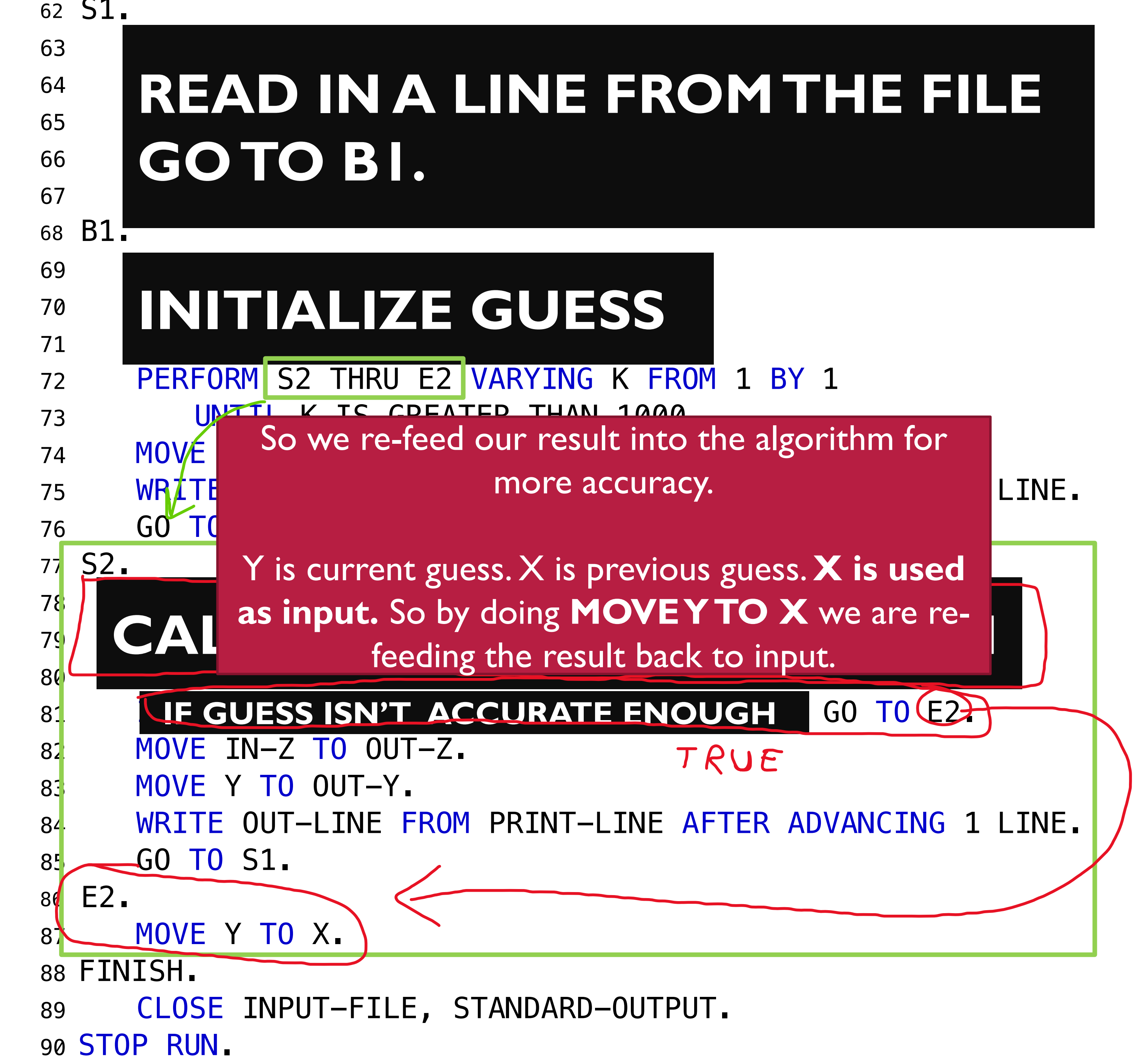
CALCULATE NEXT ITERATION

IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.

```
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO
    WRITE OUT-LI                          ANCING 1 LINE.
    GO TO S1.
S2.
    CALCULATE NEXT ITERATION
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Our guess isn't accurate enough



```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE
    WRITE                                                LINE.
    GO TO
S2.
   CAL
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

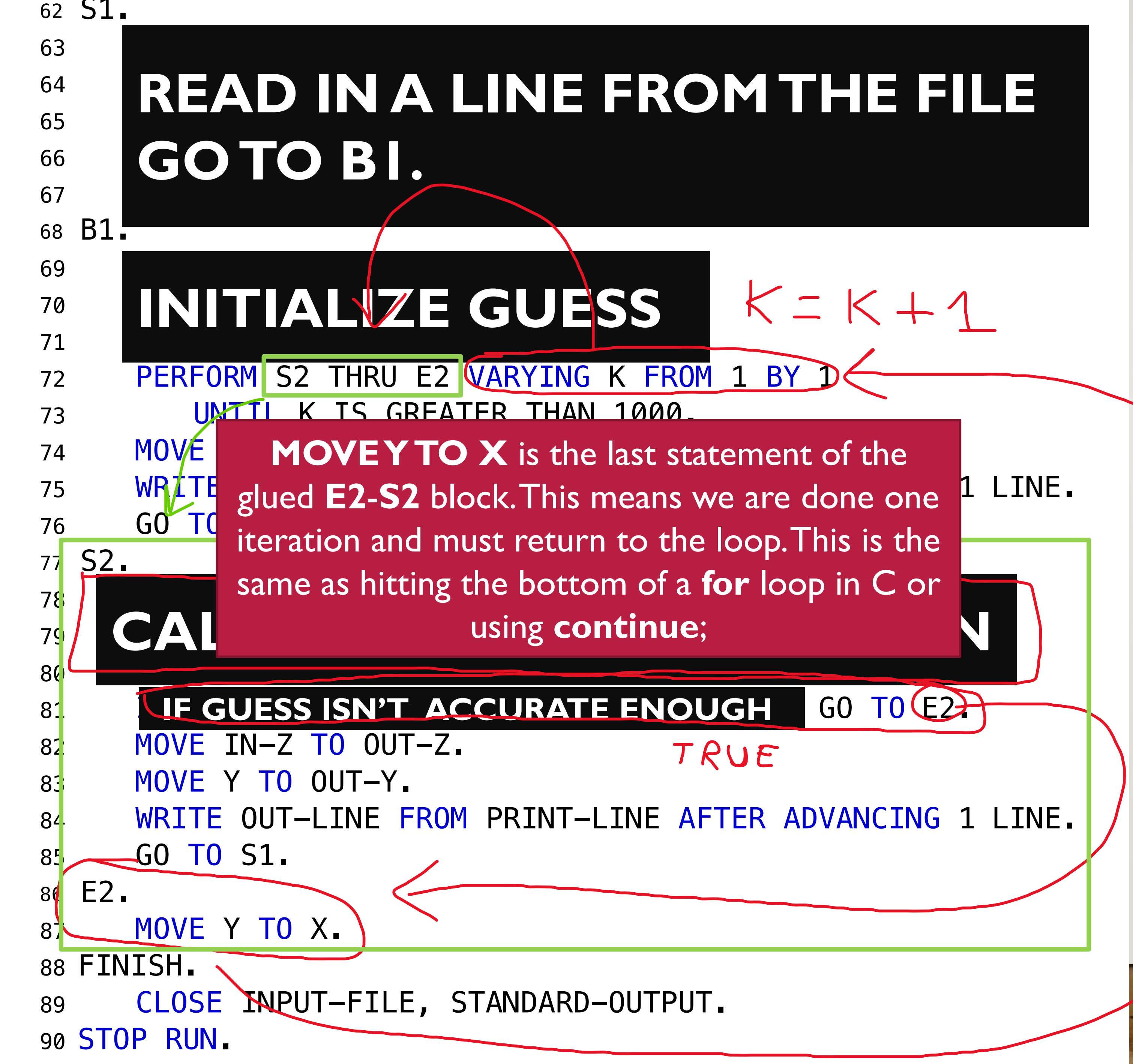
So we re-feed our result into the algorithm for more accuracy.

Y is current guess. X is previous guess. **X is used as input.** So by doing **MOVE Y TO X** we are re-feeding the result back to input.

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE
    WRITE                                             1 LINE.
    GO TO
S2.
    CAL                                           N
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

TRUE

MOVE Y TO X is the last statement of the glued **E2-S2** block. This means we are done one iteration and must return to the loop. This is the same as hitting the bottom of a **for** loop in C or using **continue**;



```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE
    WRITE                                          1 LINE.
    GO TO
S2.
 CAL                                            N
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

K = K + 1

TRUE

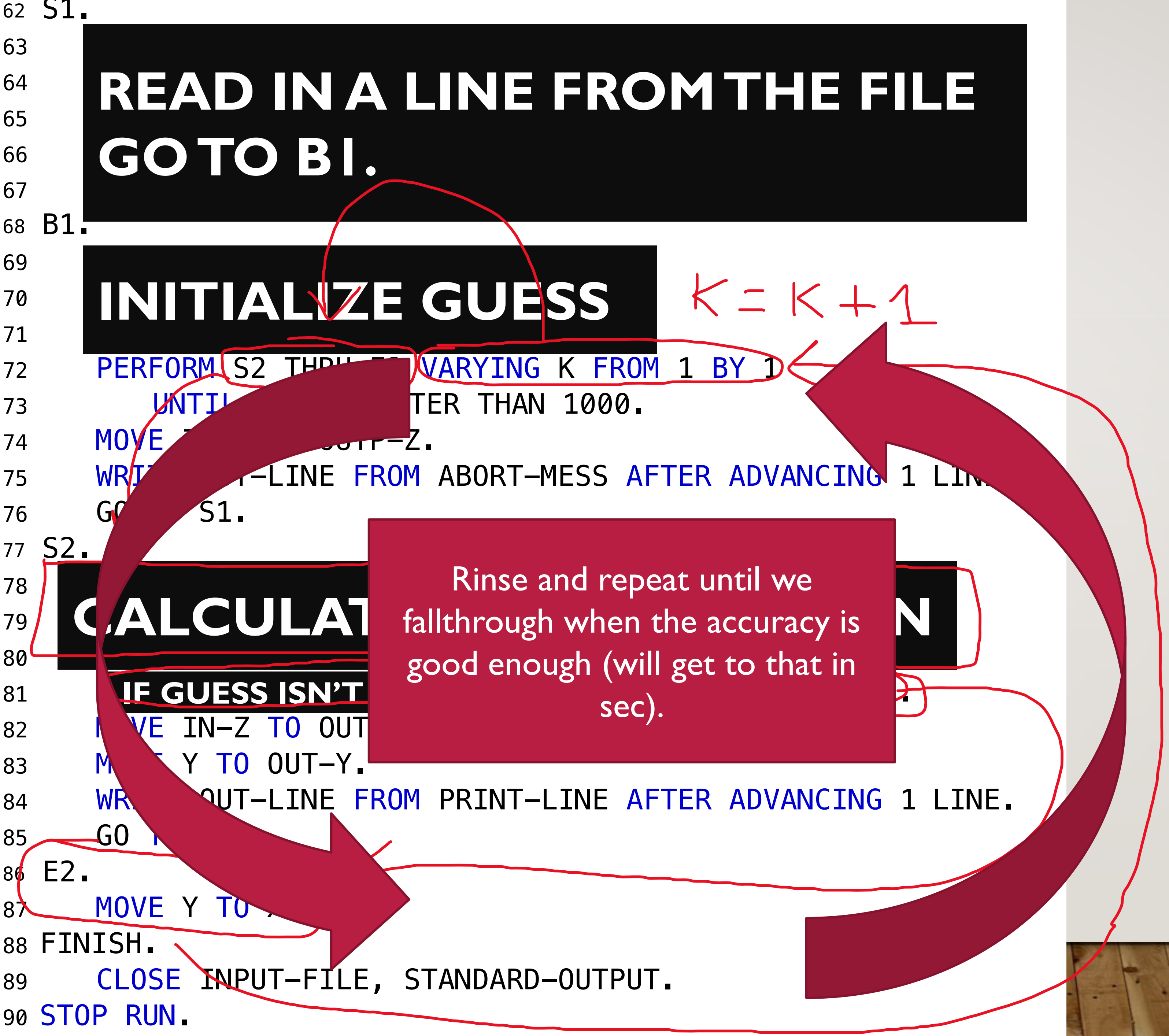
MOVE Y TO X is the last statement of the glued **E2-S2** block. This means we are done one iteration and must return to the loop. This is the same as hitting the bottom of a **for** loop in C or using **continue**;

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT                      E.
    GO TO S1.
S2.
   CALCULATE NEXT ITERATION
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

K = K + 1

TRUE

So we go back to the beginning of the loop and increment the K counting variable.



```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU    VARYING K FROM 1 BY 1
       UNTIL          TER THAN 1000.
    MOVE          OUT-Z.
    WRI    -LINE FROM ABORT-MESS AFTER ADVANCING 1 LIN
    GO    S1.
S2.
   CALCULAT                        N
     IF GUESS ISN'T
    M VE IN-Z TO OUT
    M    Y TO OUT-Y.
    WR    OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO
E2.
    MOVE Y TO
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

K = K + 1

Unless you are entering a 600 digit number (COBOL doesn't let you do that), this loop will never reach 1000. **S2** will eventually break out of the **PERFORM's scope**.

```
S1.
```

READ IN A LINE FROM THE FILE GO TO B1.

```
B1.
```

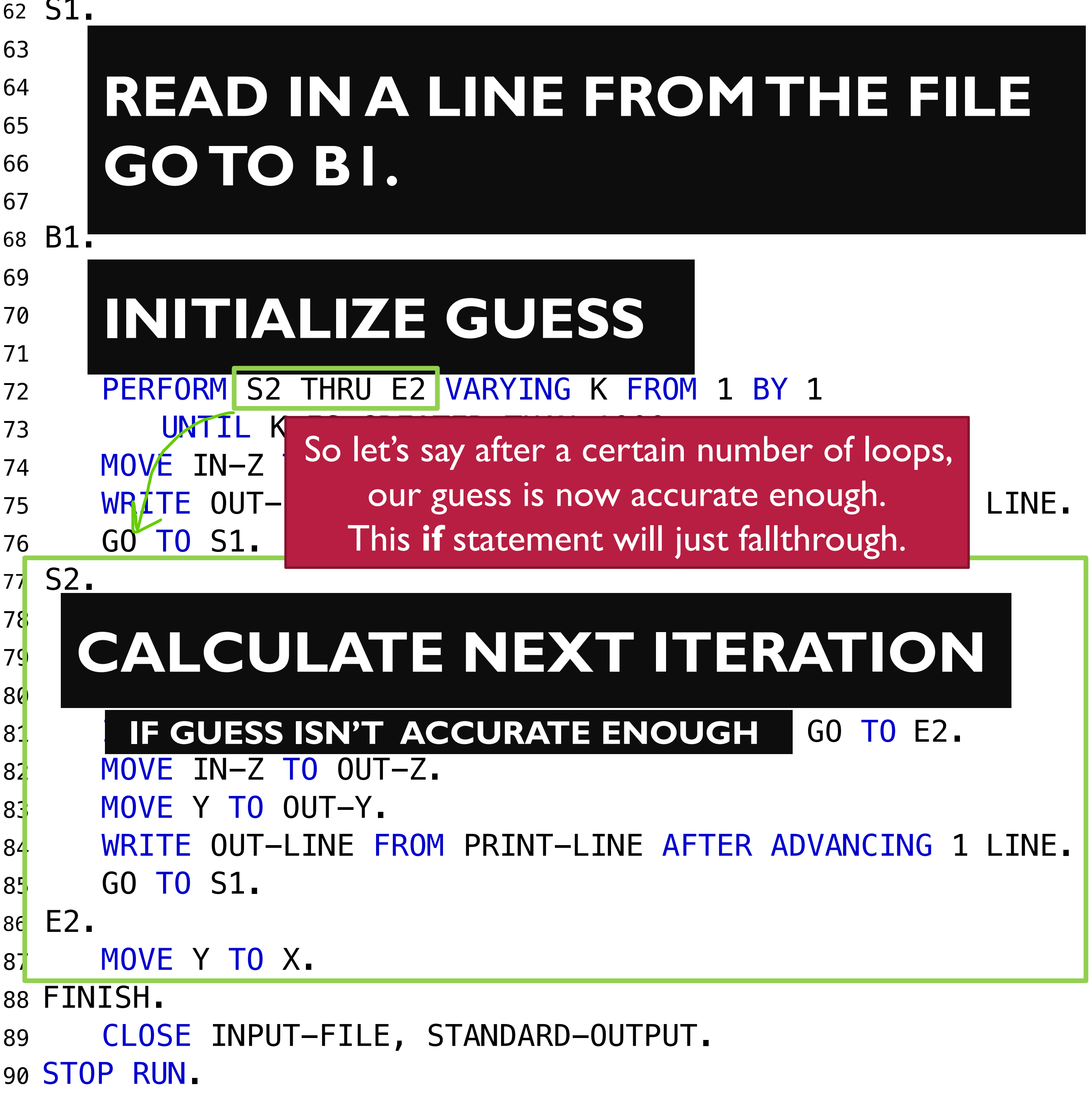
INITIALIZE GUESS

```
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
       UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1
S2.
```

Here is what that looks like.

CALCULATE NEXT ITERATION

```
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```



```
S1.
```

READ IN A LINE FROM THE FILE GO TO B1.

```
B1.
```

INITIALIZE GUESS

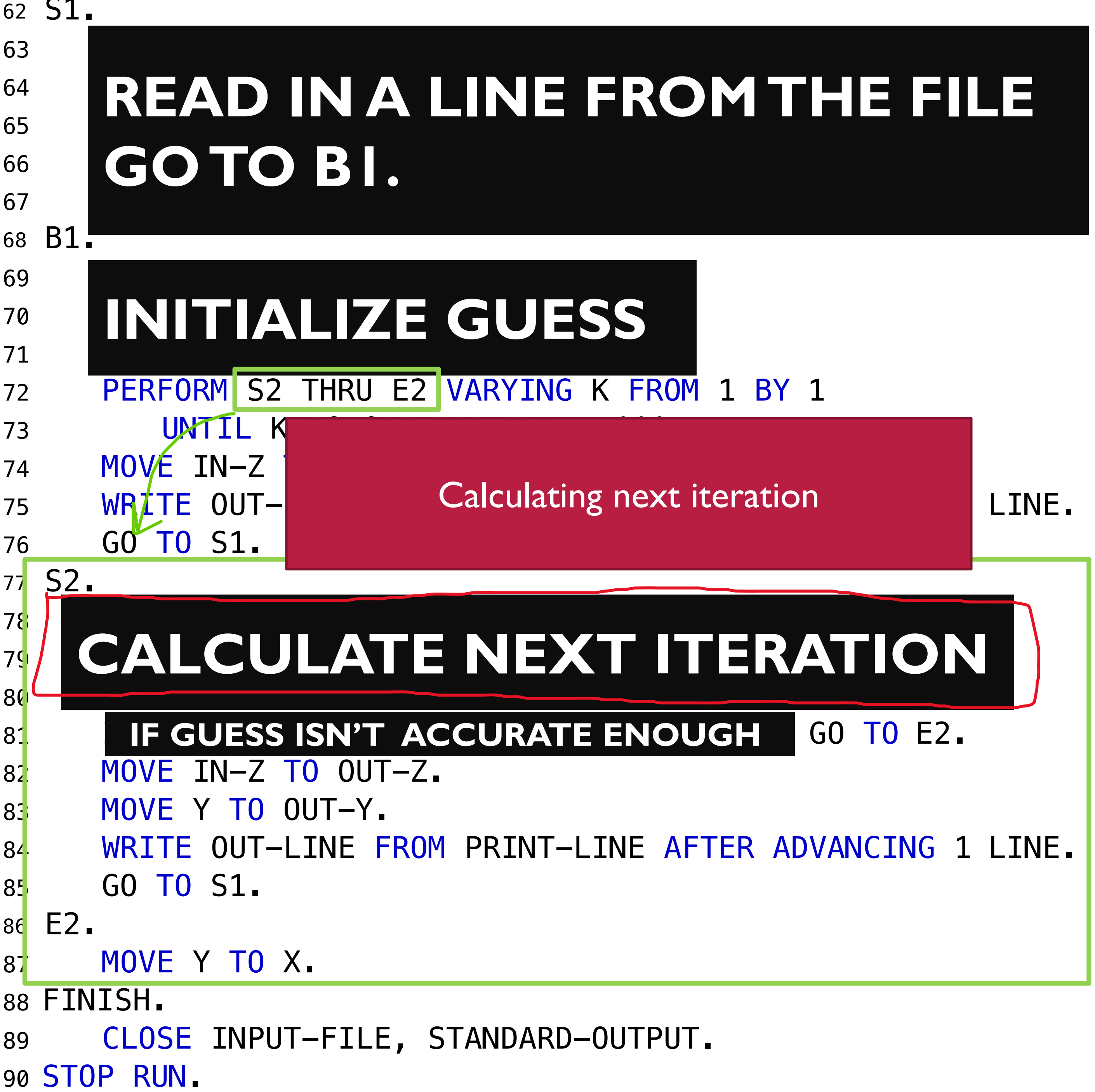
```
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K
    MOVE IN-Z
    WRITE OUT-                                         LINE.
    GO TO S1.
```

So let's say after a certain number of loops, our guess is now accurate enough. This **if** statement will just fallthrough.

```
S2.
```

CALCULATE NEXT ITERATION

```
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

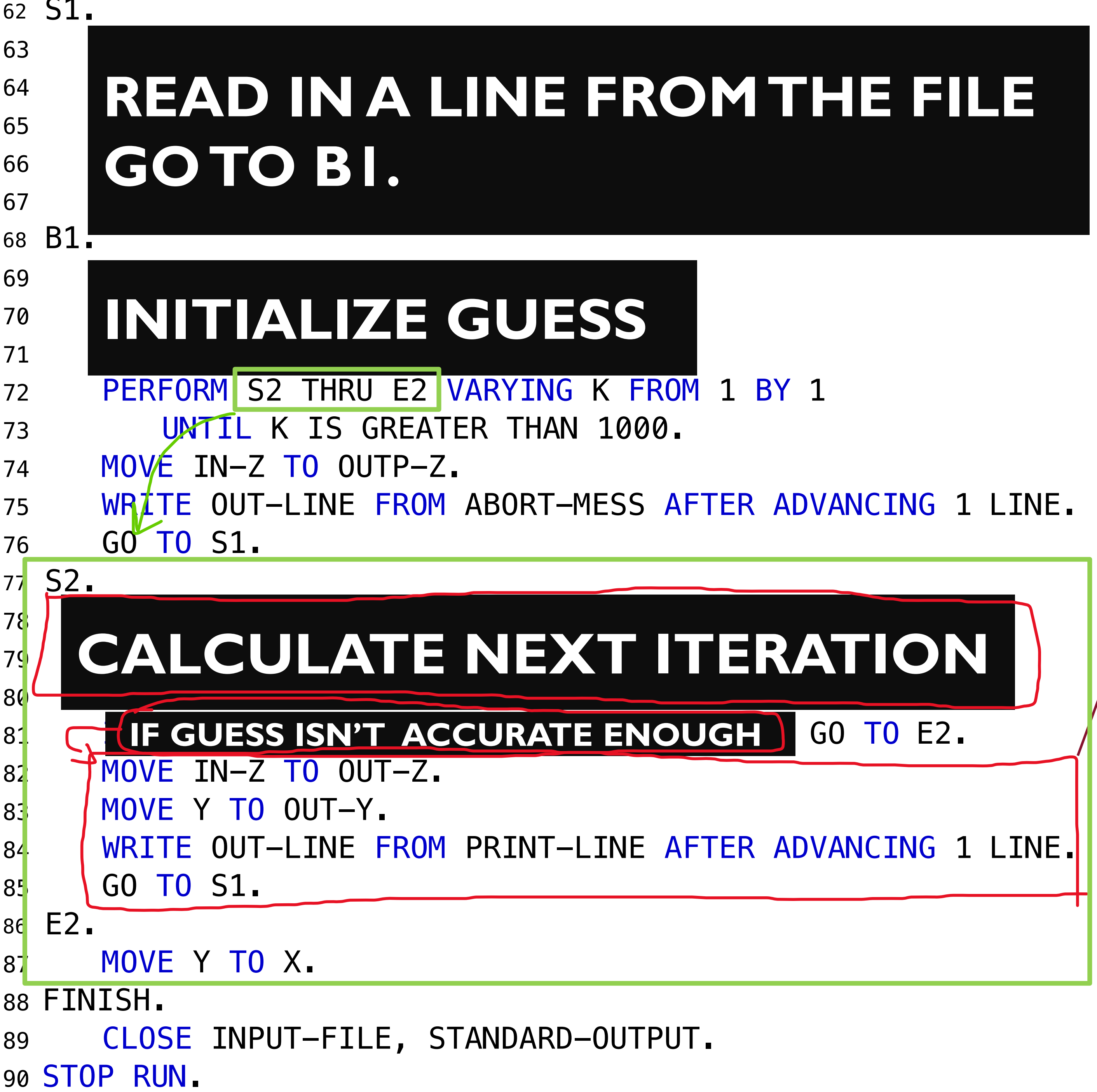


```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K
    MOVE IN-Z
    WRITE OUT-                                  LINE.
    GO TO S1.
S2.
  CALCULATE NEXT ITERATION
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Calculating next iteration

```
S1.

    READ IN A LINE FROM THE FILE
    GO TO B1.

B1.

    INITIALIZE GUESS

    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K
    MOVE IN-Z
    WRITE OUT-                                       LINE.
    GO TO S1.
S2.

    CALCULATE NEXT ITERATION

    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

This **if** statement doesn't execute and we fallthrough to the statements following it.



```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
  CALCULATE NEXT ITERATION
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

It prints the answer. We no longer need to jump to **E2** to re-feed.

Then it will **GO TO S1**.

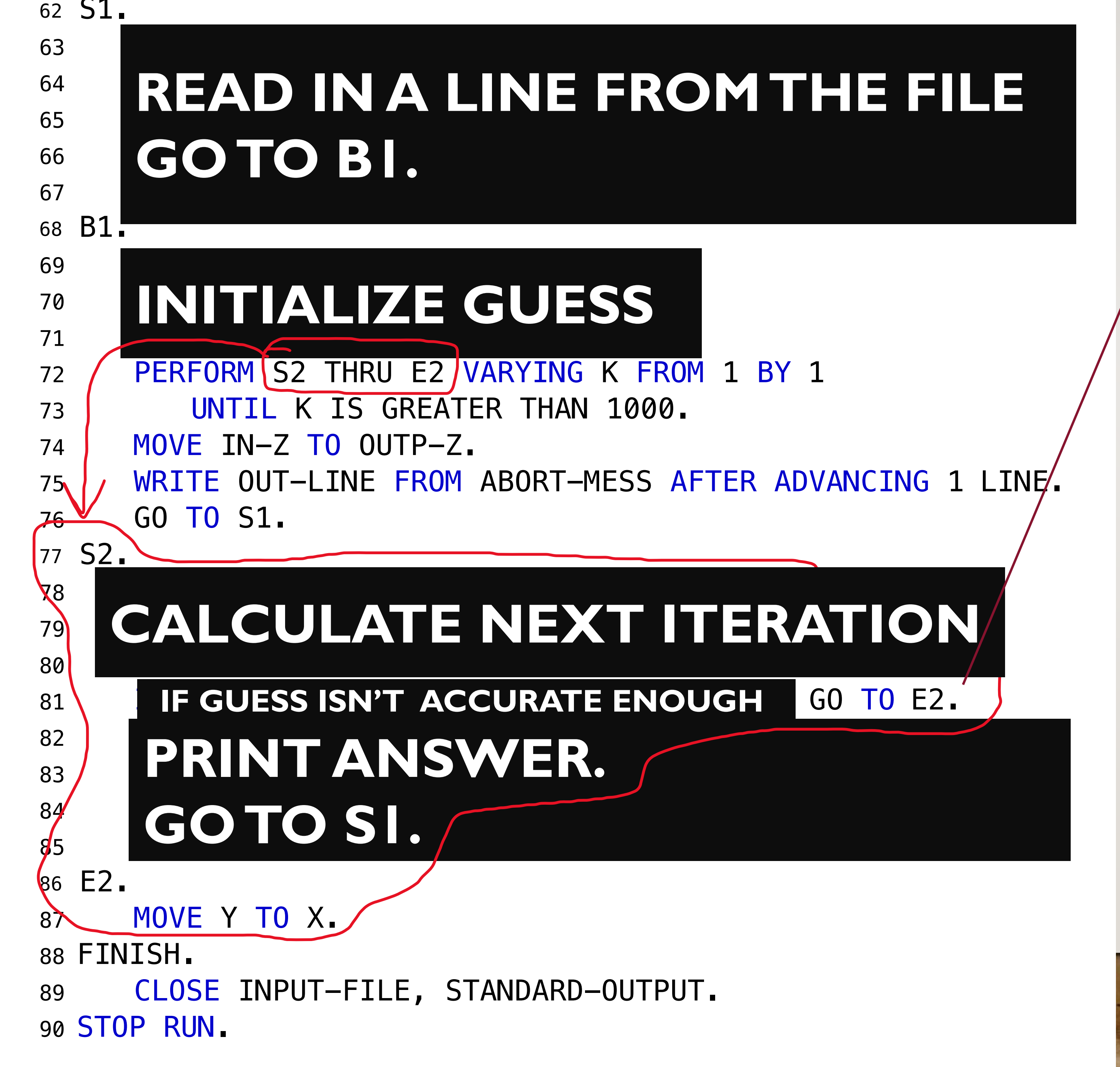
<u>This breaks the 1000 count **PERFORM** loop.</u>

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
  CALCULATE NEXT ITERATION
    IF GUESS ISN'T  ACCURATE ENOUGH GO TO E2.
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Hope you really dislike *fallthrough* logic now. Here, I'll erase that block for you.

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
   CALCULATE NEXT ITERATION
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    PRINT ANSWER.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Hope you really dislike *fallthrough* logic now. Here, I'll erase that block for you.



```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
       UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    CALCULATE NEXT ITERATION
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    PRINT ANSWER.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Recall that the **GO TO E2** statement does not break the **PERFORM** statement's scope because it stays within the **S2-E2** block.

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
   CALCULATE NEXT ITERATION
     IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    PRINT ANSWER.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Also recall that the success path calling **GO TO S1** does break the scope. Every time a correct answer is found, it ends up prematurely breaking the **PERFORM LOOP**. Why would the developer want the **PERFORM LOOP** broken?

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
       UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
   CALCULATE NEXT ITERATION
    IF GUESS ISN'T  ACCURATE ENOUGH GO TO E2.
    PRINT ANSWER.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Once again it is because of **fallthrough.** This loop is **never supposed to finish**. It was made to be broken out of using **GO TO**. I will show you why.

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
   CALCULATE NEXT ITERATION
    IF GUESS ISN'T  ACCURATE ENOUGH GO TO E2.
    PRINT ANSWER.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

This is a fallthrough for the **ABORT-MESS** or “Abort statement” that is sent when the correct level of accuracy isn’t reached in 1000 iterations.

The **PERFORM** loop *tries* to count to 1000 and hopes it doesn’t actually reach it, but if it does, it falls through and prints that message because after 1000 attempts, it aborts.

Let’s abstract and black-box that away.

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
   CALCULATE NEXT ITERATION
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    PRINT ANSWER.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Begone!!!

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
    CALCULATE NEXT ITERATION
    IF GUESS ISN’T ACCURATE ENOUGH GO TO E2.
    PRINT ANSWER.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

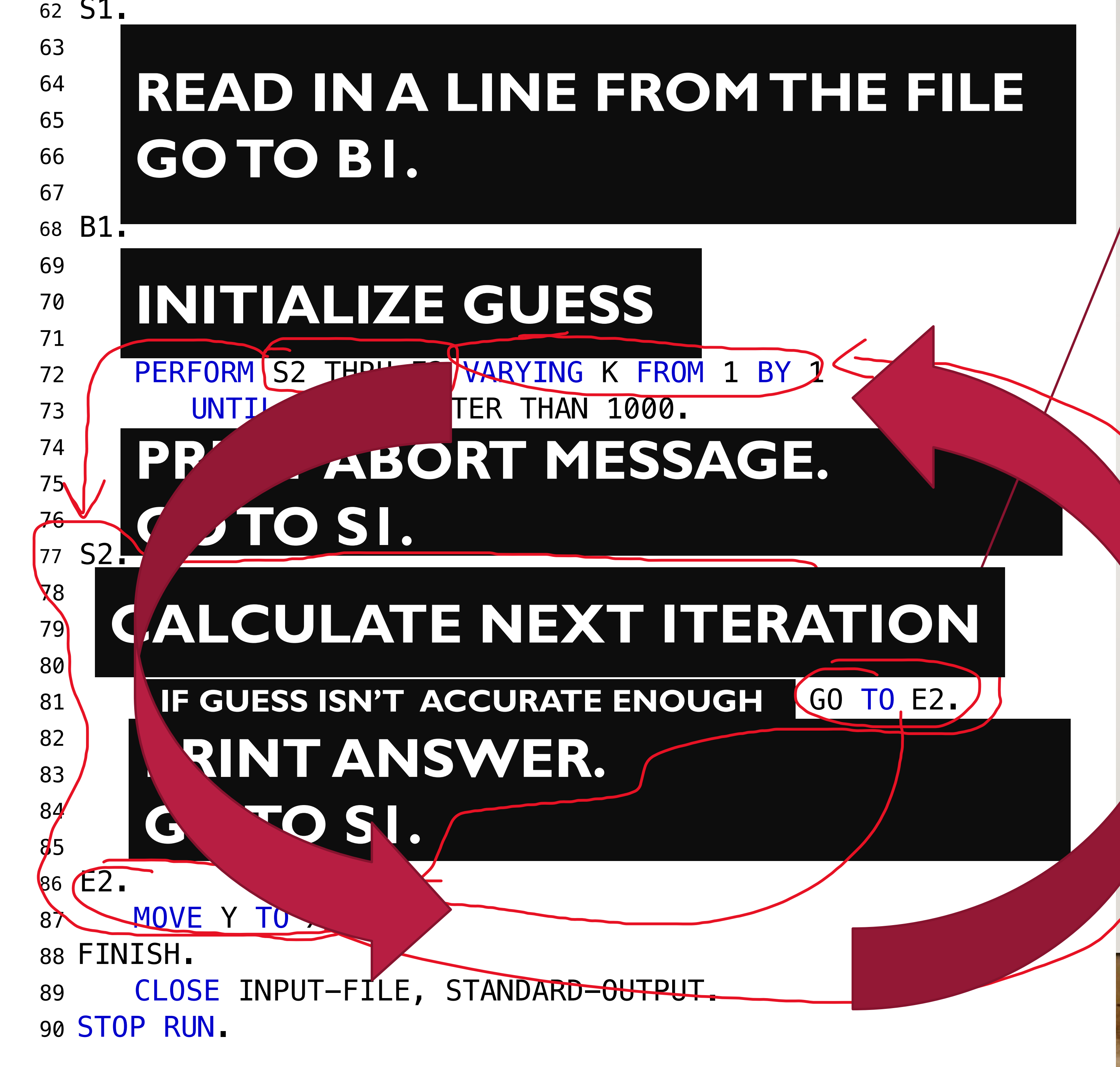
So, if the **PERFORM** loop was never meant to reach this ABORT statement through normal operation…

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
    CALCULATE NEXT ITERATION
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    PRINT ANSWER.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

And if **E2** is just for repeated iterations to get more accuracy…

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO BI.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO SI.
S2.
    CALCULATE NEXT ITERATION
    IF GUESS ISN’T ACCURATE ENOUGH GO TO E2.
    PRINT ANSWER.
    GO TO SI.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

And if **E2** is just for repeated iterations to get more accuracy…



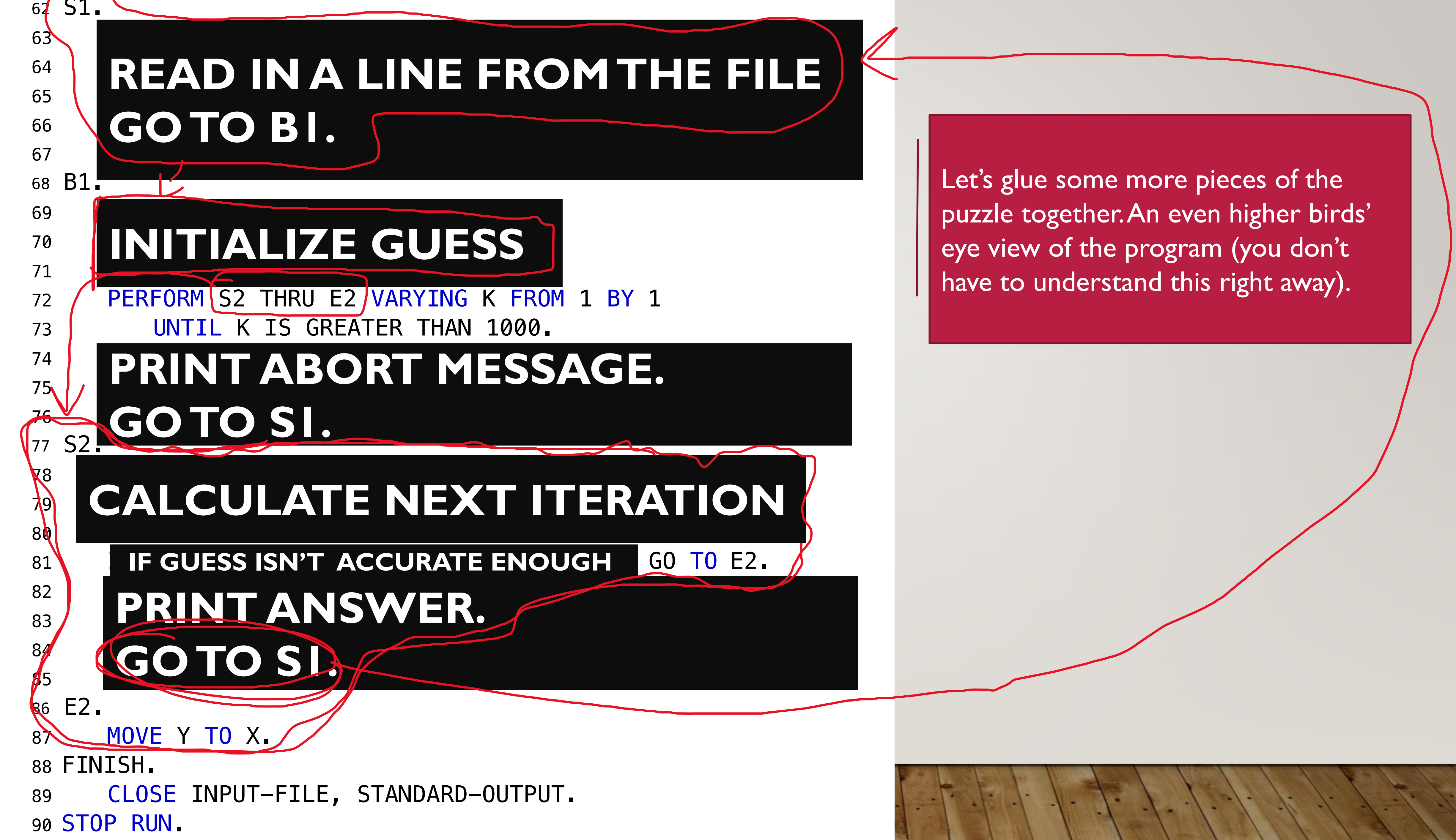
```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
   CALCULATE NEXT ITERATION
     IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    PRINT ANSWER.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

And if **E2** is just for repeated iterations to get more accuracy...

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
   CALCULATE NEXT ITERATION
    IF GUESS ISN’T ACCURATE ENOUGH GO TO E2.
    PRINT ANSWER.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

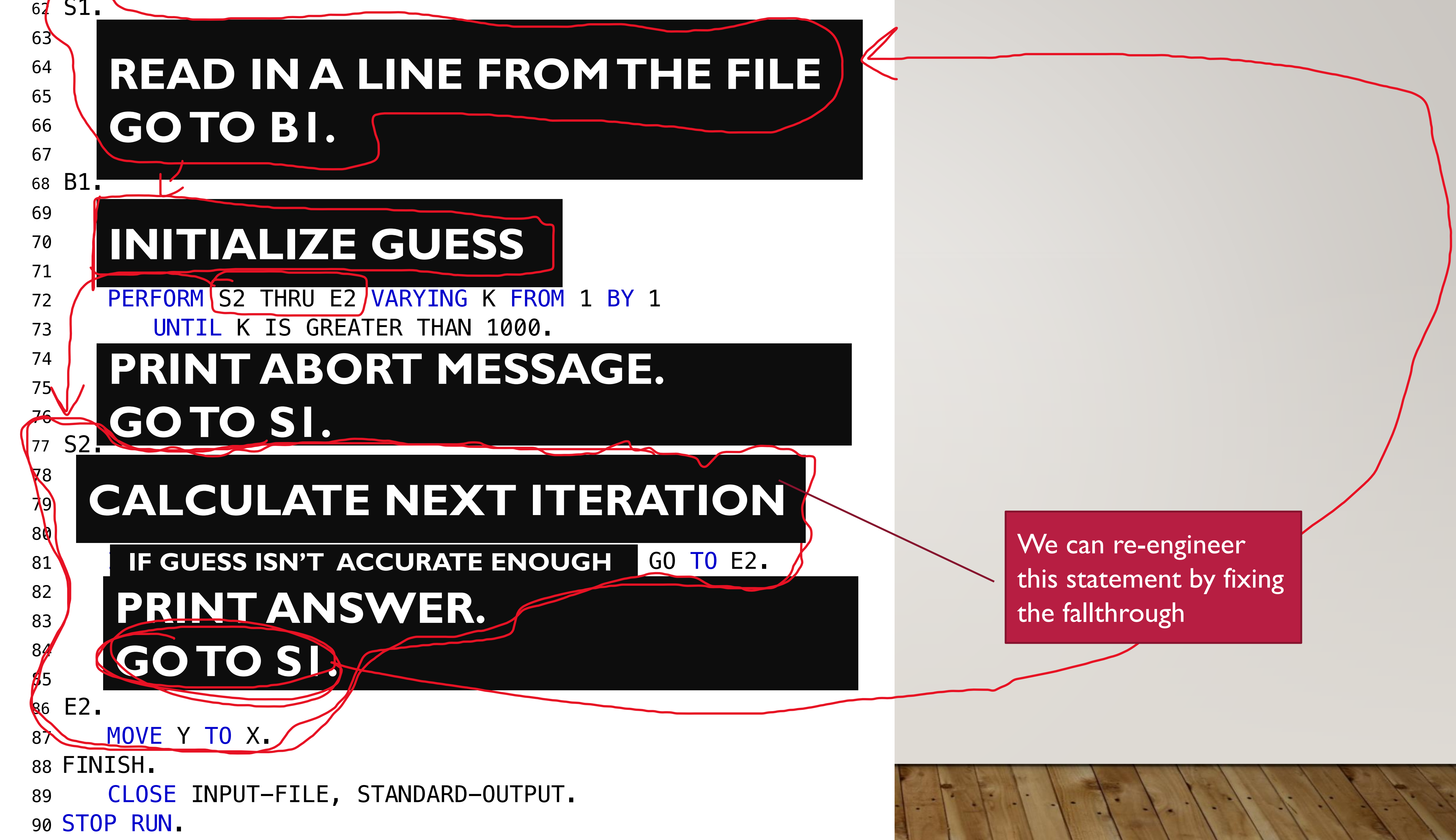
Then this must be the true goal of the **PERFORM** statement!

To continuously trigger the **GO TO E2** until we are accurate enough to reach the **GO TO S1** statement.



```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
    CALCULATE NEXT ITERATION
    IF GUESS ISN'T  ACCURATE ENOUGH GO TO E2.
    PRINT ANSWER.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Let's glue some more pieces of the puzzle together. An even higher birds' eye view of the program (you don't have to understand this right away).

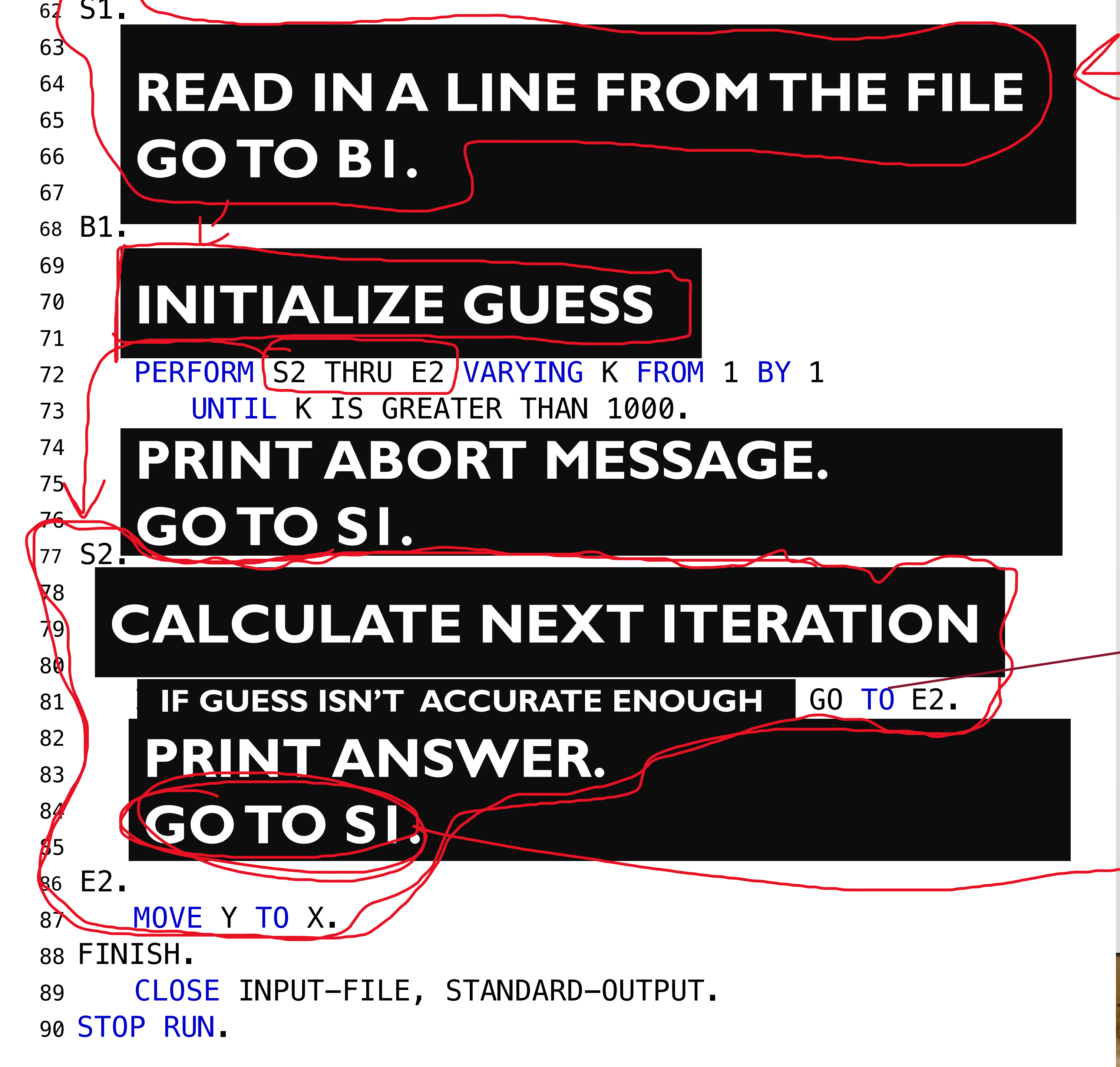


```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
   CALCULATE NEXT ITERATION
     IF GUESS ISN’T ACCURATE ENOUGH GO TO E2.
     PRINT ANSWER.
     GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

We can re-engineer this statement by fixing the fallthrough

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
    CALCULATE NEXT ITERATION
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    PRINT ANSWER.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

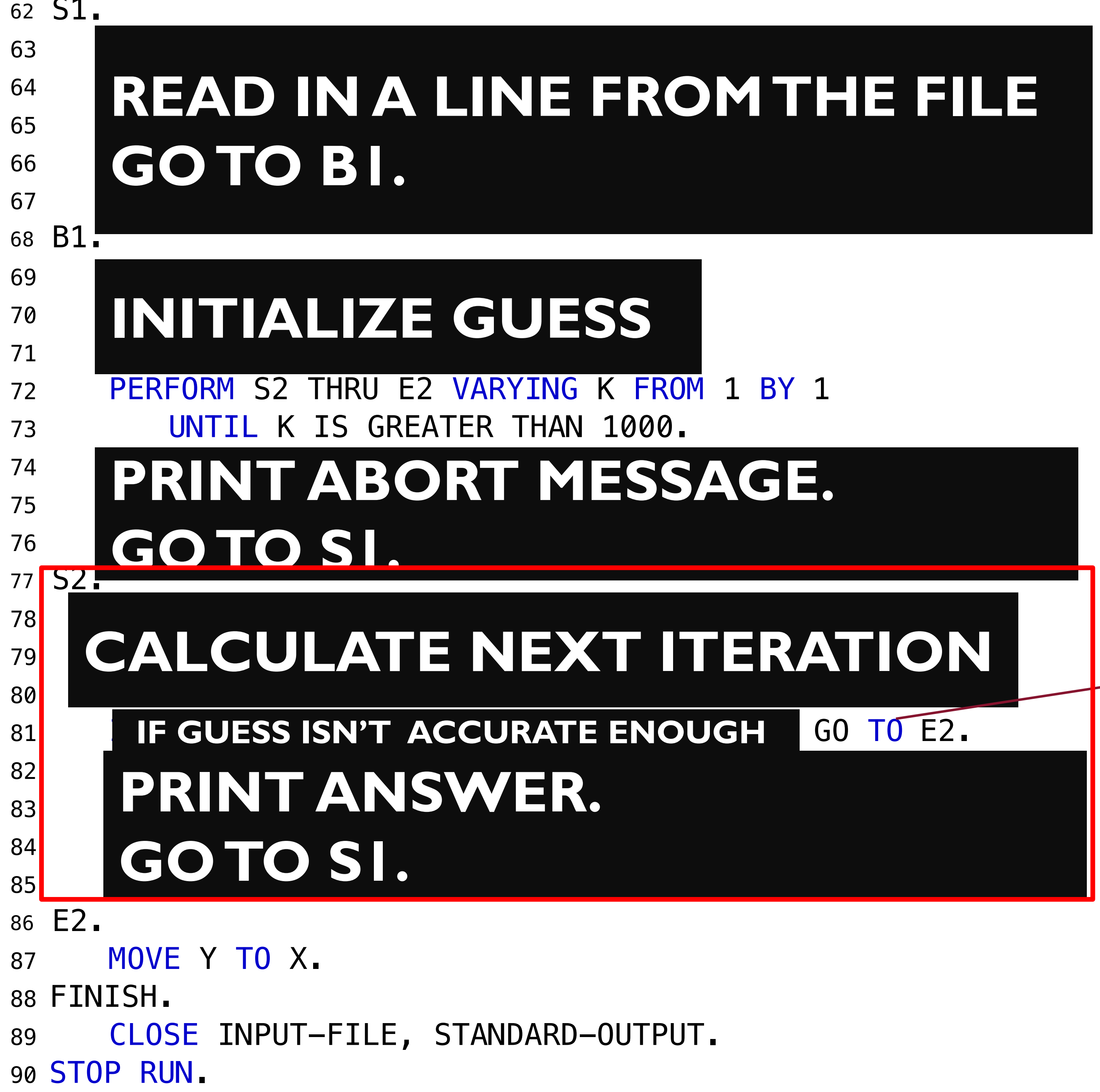
Recall that the fallthrough is an implied if-else that gives up control after you evaluate the condition,



```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
   CALCULATE NEXT ITERATION
    IF GUESS ISN'T  ACCURATE ENOUGH GO TO E2.
    PRINT ANSWER.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

You can think of this statement as:

```
IF GUESS ISN'T ACCURATE ENOUGH
        GO TO E2
ELSE
        PRINT ANSWER
        GO TO S1
END-IF
```



```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
   CALCULATE NEXT ITERATION
    IF GUESS ISN'T ACCURATE ENOUGH GO TO E2.
    PRINT ANSWER.
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

You can think of this statement as:

```
IF GUESS ISN'T ACCURATE ENOUGH
    GO TO E2
ELSE
    PRINT ANSWER
    GO TO S1
END-IF
```

```
S1.
```

READ IN A LINE FROM THE FILE
GO TO B1.

```
B1.
```

INITIALIZE GUESS

```
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
```

PRINT ABORT MESSAGE.
GO TO S1.

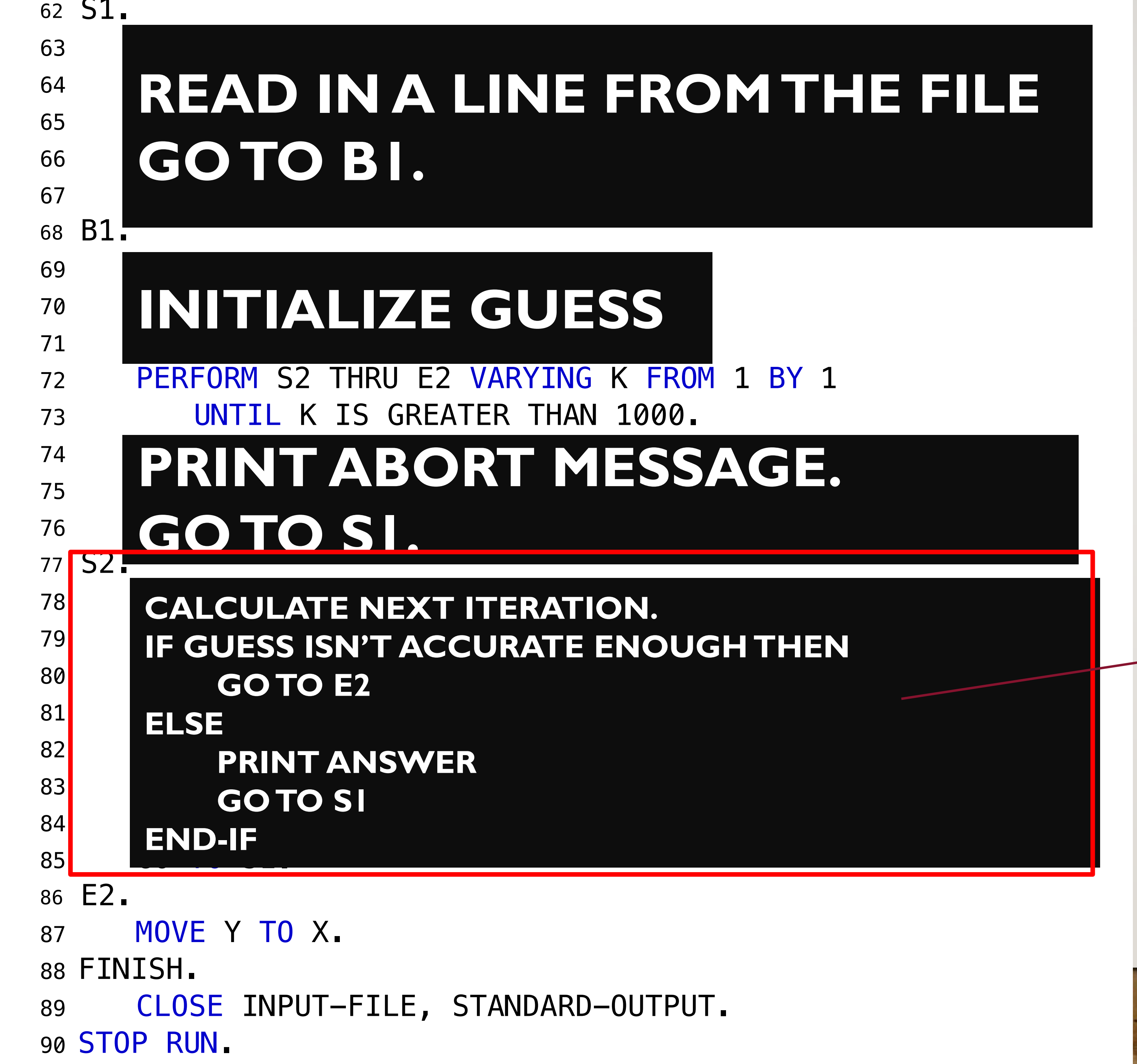
```
S2.
```

CALCULATE NEXT ITERATION.
IF GUESS ISN'T ACCURATE ENOUGH GO TO E2
PRINT ANSWER
GO TO S1

```
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

You can think of this statement as:

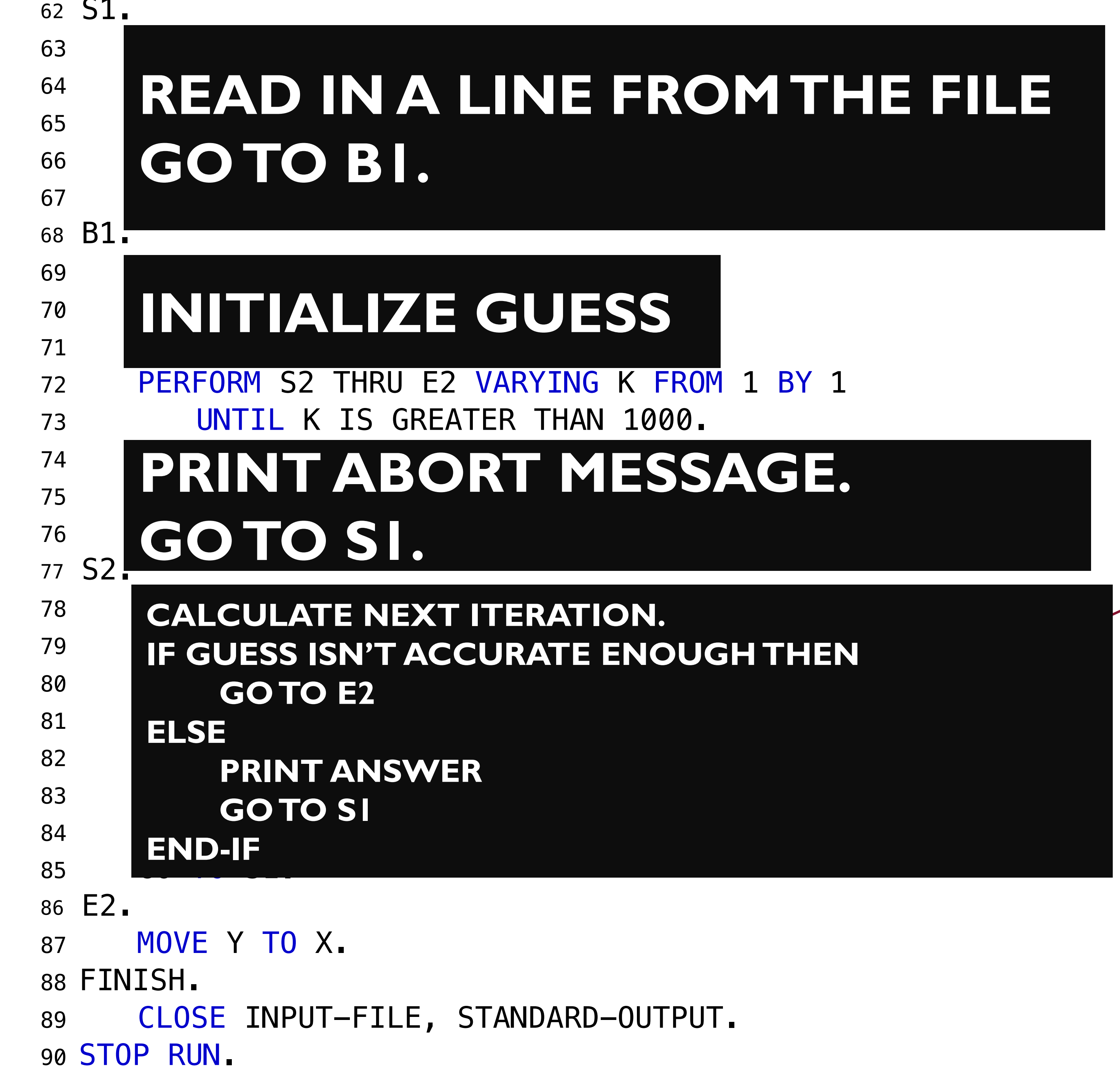
```
IF GUESS ISN'T ACCURATE ENOUGH
      GO TO E2
ELSE
      PRINT ANSWER
      GO TO S1
END-IF
```



```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
    CALCULATE NEXT ITERATION.
    IF GUESS ISN'T ACCURATE ENOUGH THEN
        GO TO E2
    ELSE
        PRINT ANSWER
        GO TO S1
    END-IF
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

You can think of this statement as:

```
IF GUESS ISN'T ACCURATE ENOUGH
        GO TO E2
ELSE
        PRINT ANSWER
        GO TO S1
END-IF
```

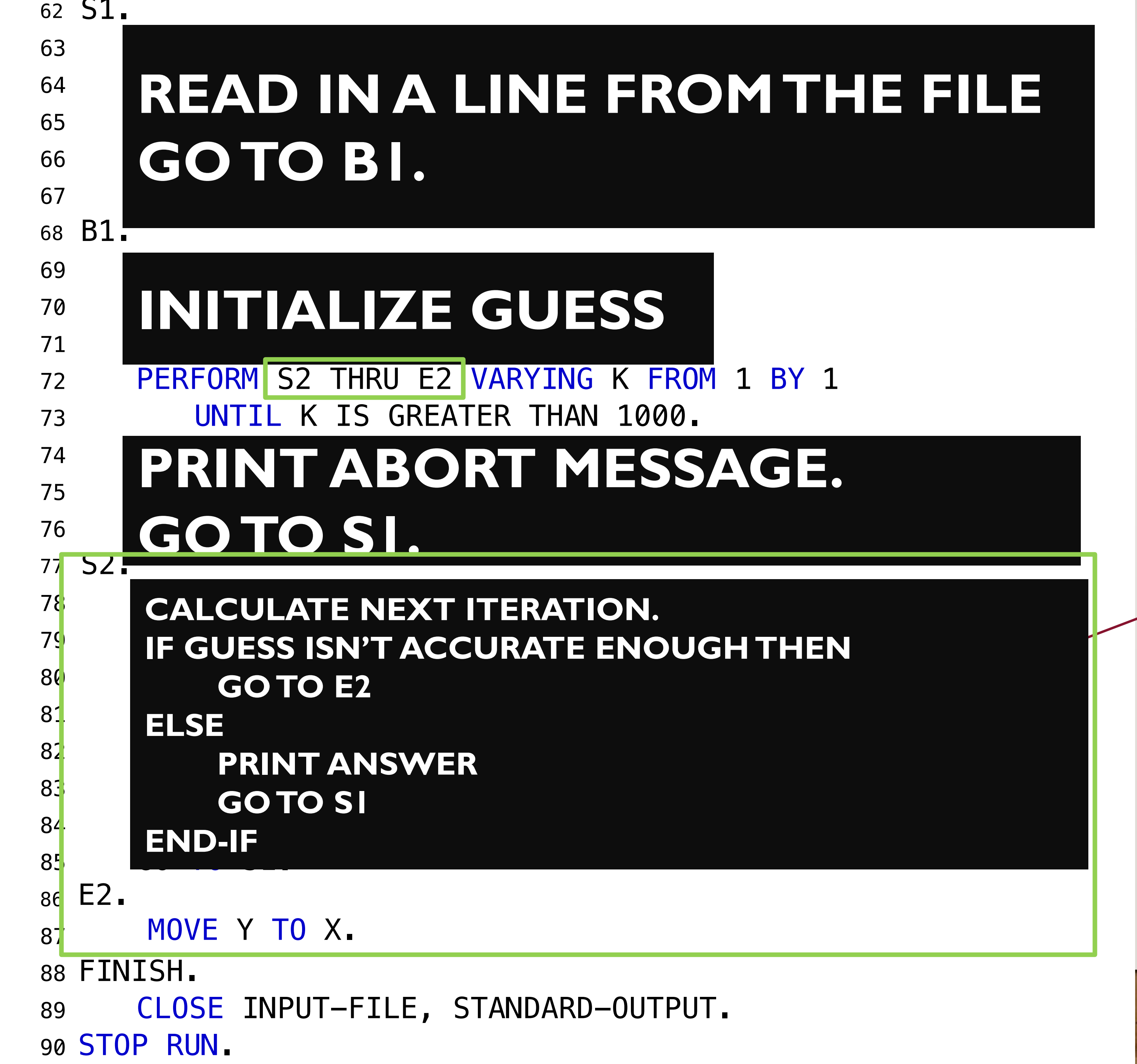


```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
    CALCULATE NEXT ITERATION.
    IF GUESS ISN'T ACCURATE ENOUGH THEN
        GO TO E2
    ELSE
        PRINT ANSWER
        GO TO S1
    END-IF
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

This is what S2 can be converted to. It's not that much of a step up, but it's honest work.

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
    CALCULATE NEXT ITERATION.
    IF GUESS ISN’T ACCURATE ENOUGH THEN
        GO TO E2
    ELSE
        PRINT ANSWER
        GO TO S1
    END-IF
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

So, the crux of this presentation: how can we remove that blasted **E2**?



```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
    CALCULATE NEXT ITERATION.
    IF GUESS ISN'T ACCURATE ENOUGH THEN
        GO TO E2
    ELSE
        PRINT ANSWER
        GO TO S1
    END-IF
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Recall the big **S2-E2** blob.

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
    CALCULATE NEXT ITERATION.
    IF GUESS ISN'T ACCURATE ENOUGH THEN
        GO TO E2
    ELSE
        PRINT ANSWER
        GO TO S1
    END-IF
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Let's move **E2** into the **IF**, as it is only called once in **S2**.

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
    CALCULATE NEXT ITERATION.
    IF GUESS ISN'T ACCURATE ENOUGH THEN
        MOVE Y TO X
    ELSE
        PRINT ANSWER
        GO TO S1
    END-IF
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

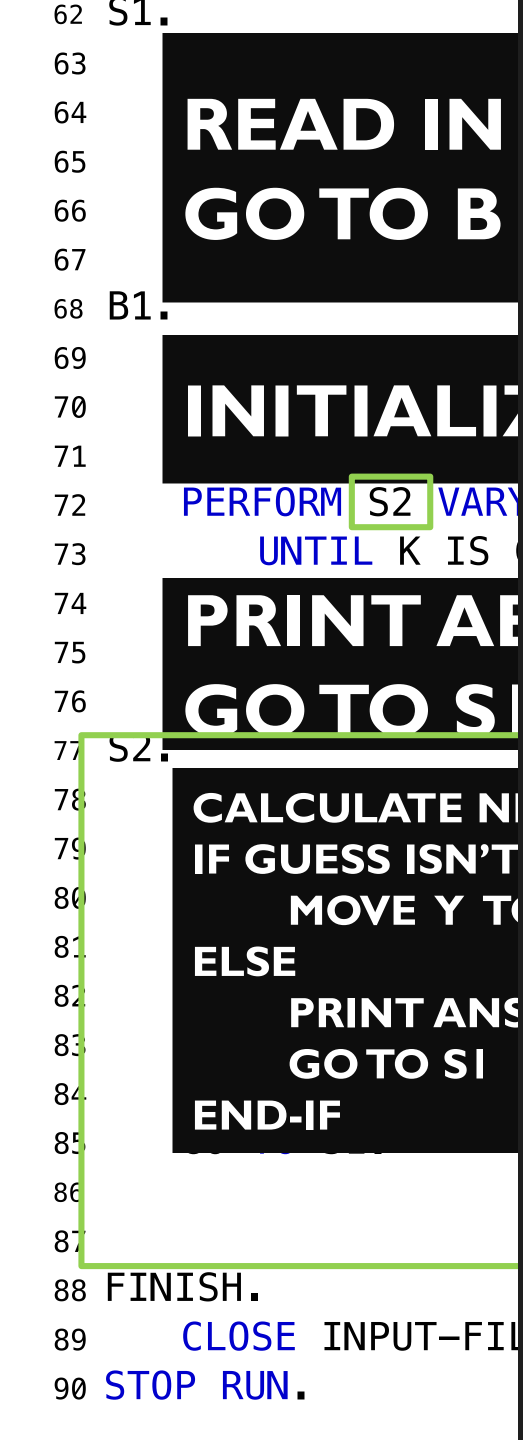
Let's move **E2** into the **IF**, as it is only called once in **S2**.

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
    CALCULATE NEXT ITERATION.
    IF GUESS ISN'T ACCURATE ENOUGH THEN
        MOVE Y TO X
    ELSE
        PRINT ANSWER
        GO TO S1
    END-IF
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

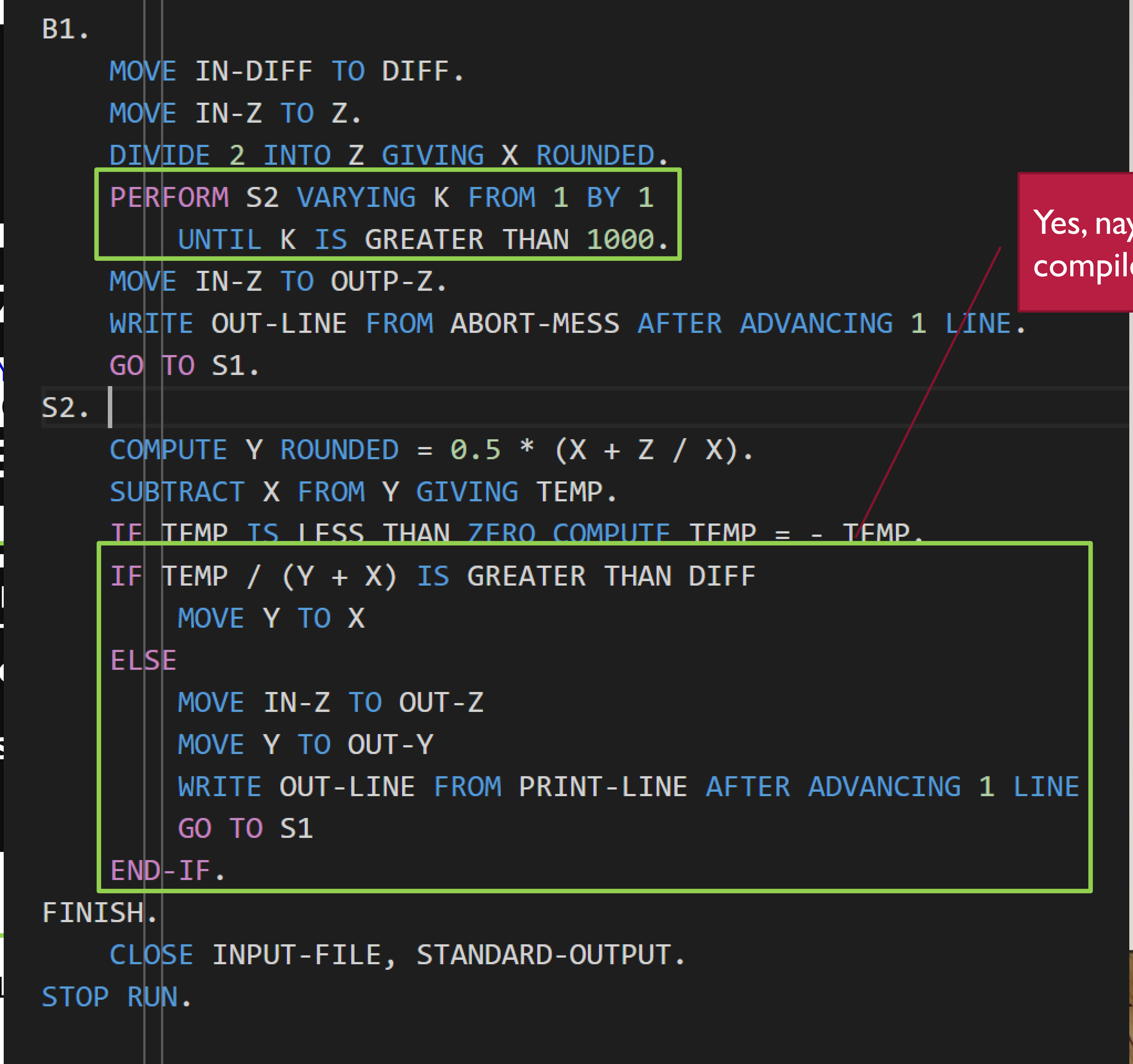
Now we can get rid of the "glued **S2 E2**" paradigm

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 VARYING K FROM 1 BY 1
       UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
     CALCULATE NEXT ITERATION.
     IF GUESS ISN'T ACCURATE ENOUGH THEN
            MOVE Y TO X
     ELSE
            PRINT ANSWER
            GO TO S1
     END-IF
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Now we can get rid of the "glued **S2 E2**" paradigm

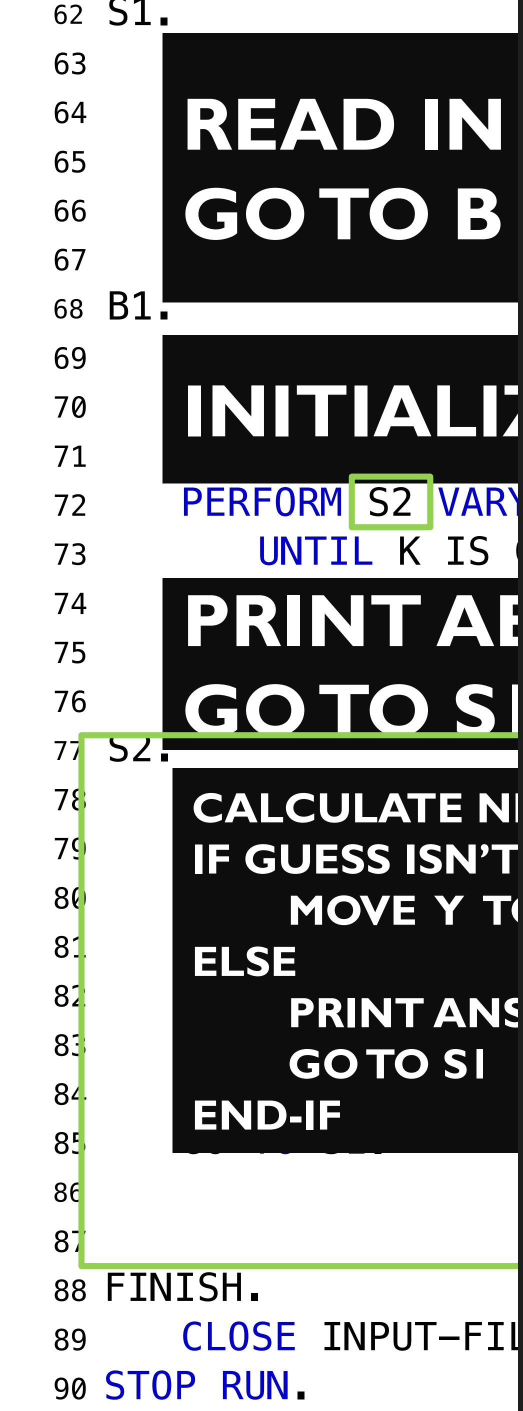


```
S1.
    READ IN
    GO TO B
B1.
    INITIALIZ
    PERFORM S2 VARY
        UNTIL K IS (
    PRINT AB
    GO TO S1
S2.
    CALCULATE N
    IF GUESS ISN'T
        MOVE Y T
    ELSE
        PRINT ANS
        GO TO S1
    END-IF
FINISH.
    CLOSE INPUT-FIL
STOP RUN.
```

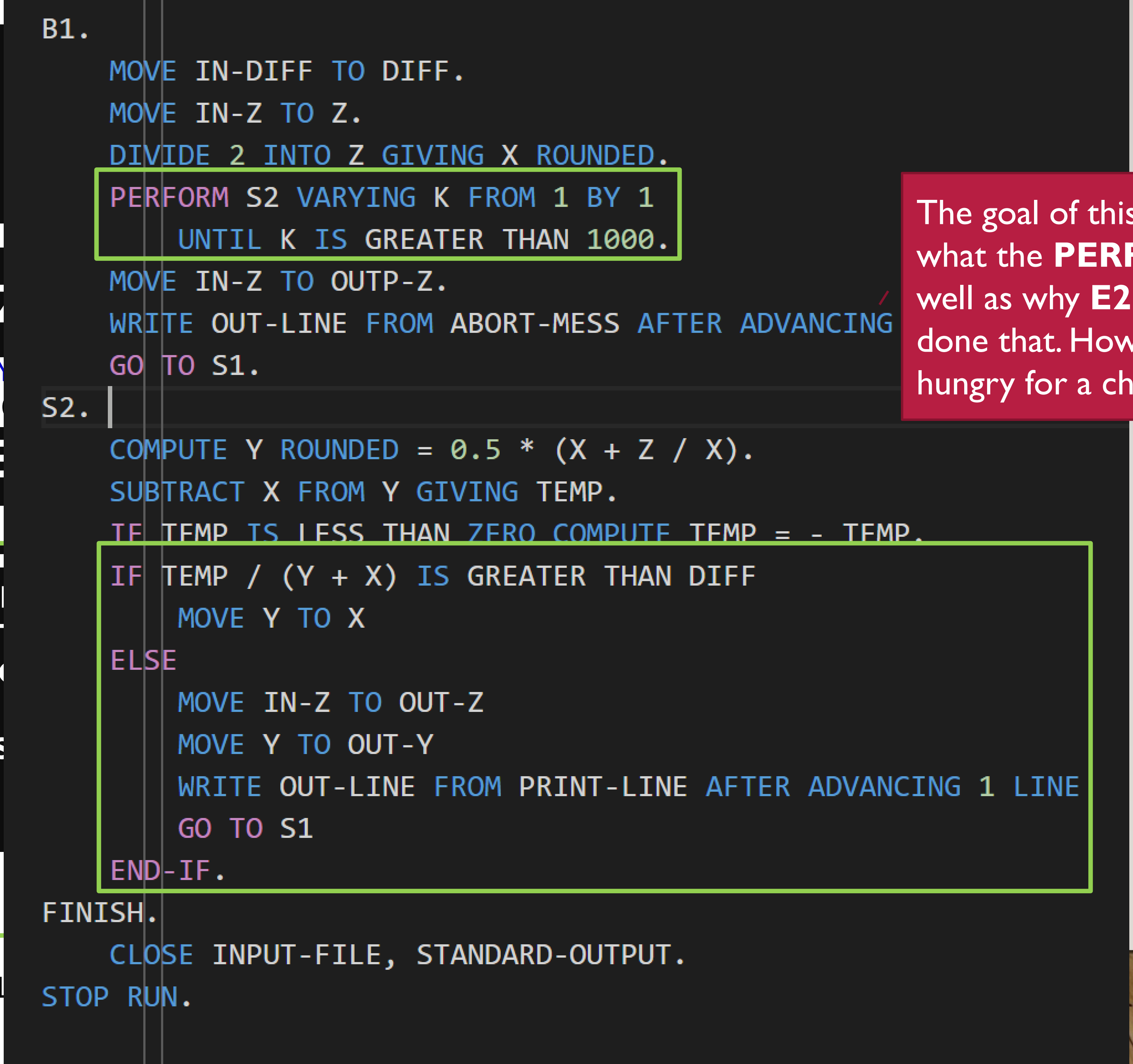


```
B1.
    MOVE IN-DIFF TO DIFF.
    MOVE IN-Z TO Z.
    DIVIDE 2 INTO Z GIVING X ROUNDED.
    PERFORM S2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF TEMP / (Y + X) IS GREATER THAN DIFF
        MOVE Y TO X
    ELSE
        MOVE IN-Z TO OUT-Z
        MOVE Y TO OUT-Y
        WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE
        GO TO S1
    END-IF.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Yes, naysayers—this will compile and run



```
S1.
    READ IN
    GO TO B
B1.
    INITIALIZ
    PERFORM S2 VARY
        UNTIL K IS (
    PRINT AE
    GO TO SI
S2.
    CALCULATE NI
    IF GUESS ISN'T
        MOVE Y T(
    ELSE
        PRINT ANS
        GO TO S1
    END-IF
FINISH.
    CLOSE INPUT-FIL
STOP RUN.
```

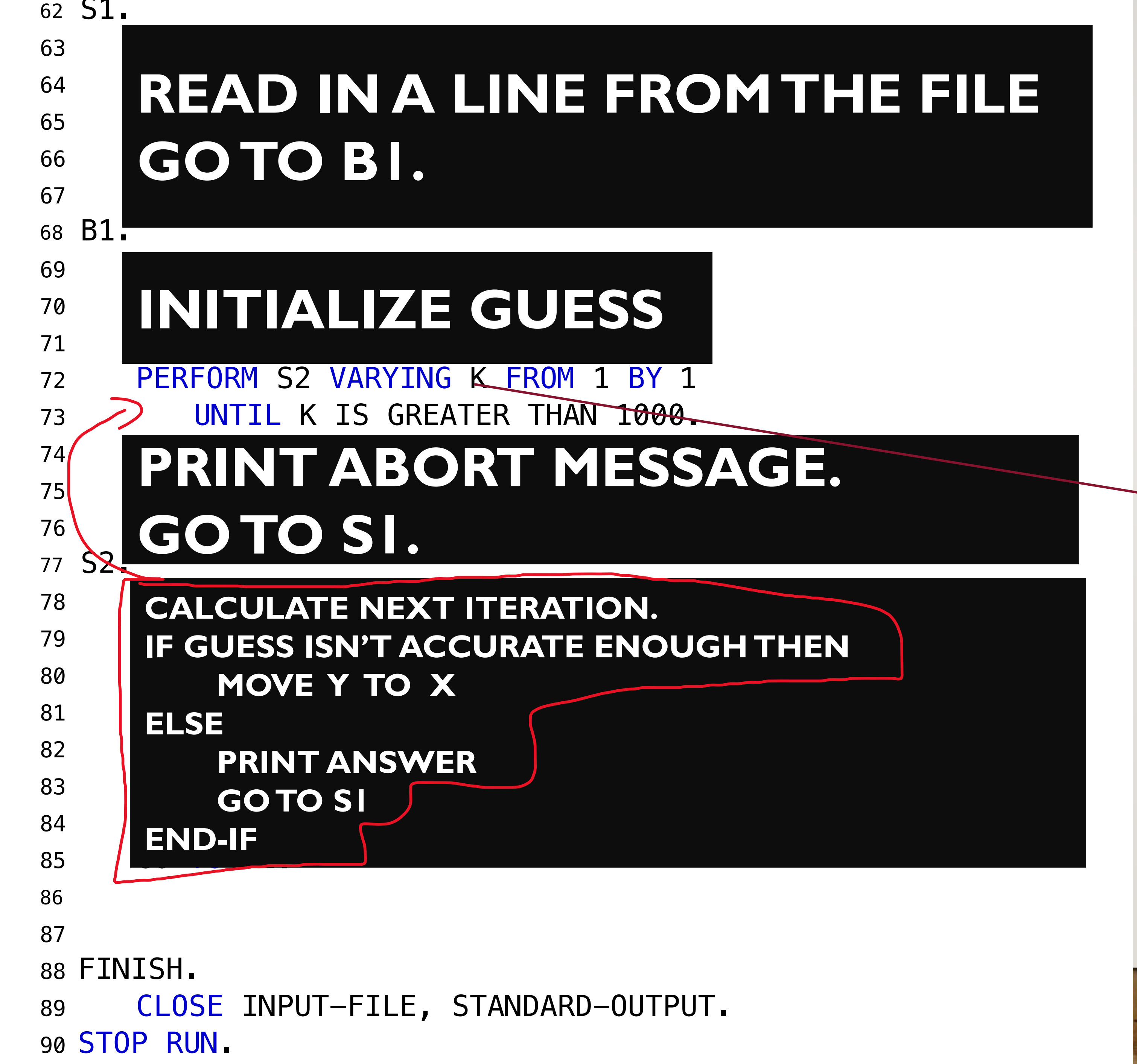


```
B1.
    MOVE IN-DIFF TO DIFF.
    MOVE IN-Z TO Z.
    DIVIDE 2 INTO Z GIVING X ROUNDED.
    PERFORM S2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    MOVE IN-Z TO OUTP-Z.
    WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING
    GO TO S1.
S2.
    COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
    SUBTRACT X FROM Y GIVING TEMP.
    IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
    IF TEMP / (Y + X) IS GREATER THAN DIFF
        MOVE Y TO X
    ELSE
        MOVE IN-Z TO OUT-Z
        MOVE Y TO OUT-Y
        WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE
        GO TO S1
    END-IF.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

The goal of this PPT is to understand what the **PERFORM** loop does, as well as why **E2** is such a bitch.I am done that. However if you are still hungry for a challenge…

```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 VARYING K FROM 1 BY 1
       UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
    CALCULATE NEXT ITERATION.
    IF GUESS ISN'T ACCURATE ENOUGH THEN
        MOVE Y TO X
    ELSE
        PRINT ANSWER
        GO TO S1
    END-IF


FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

What's next?



```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 VARYING K FROM 1 BY 1
       UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
    CALCULATE NEXT ITERATION.
    IF GUESS ISN’T ACCURATE ENOUGH THEN
        MOVE Y TO X
    ELSE
        PRINT ANSWER
        GO TO S1
    END-IF


FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

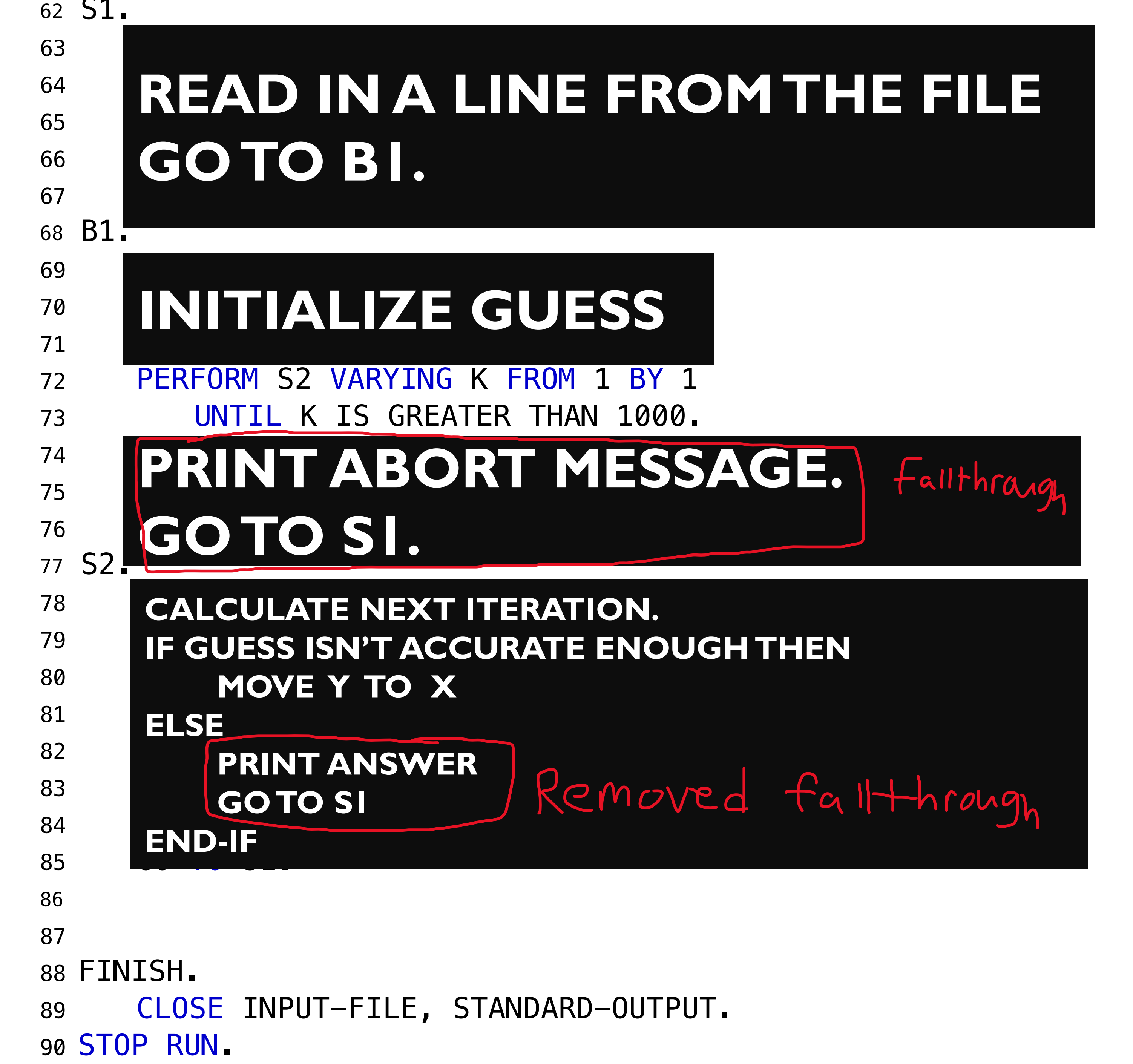
You could integrate **S2** into the **PERFORM** loop in **B1**.

Instead of:

```
PERFORM PARA.

PARA.
   DISPLAY “HELLO”
```

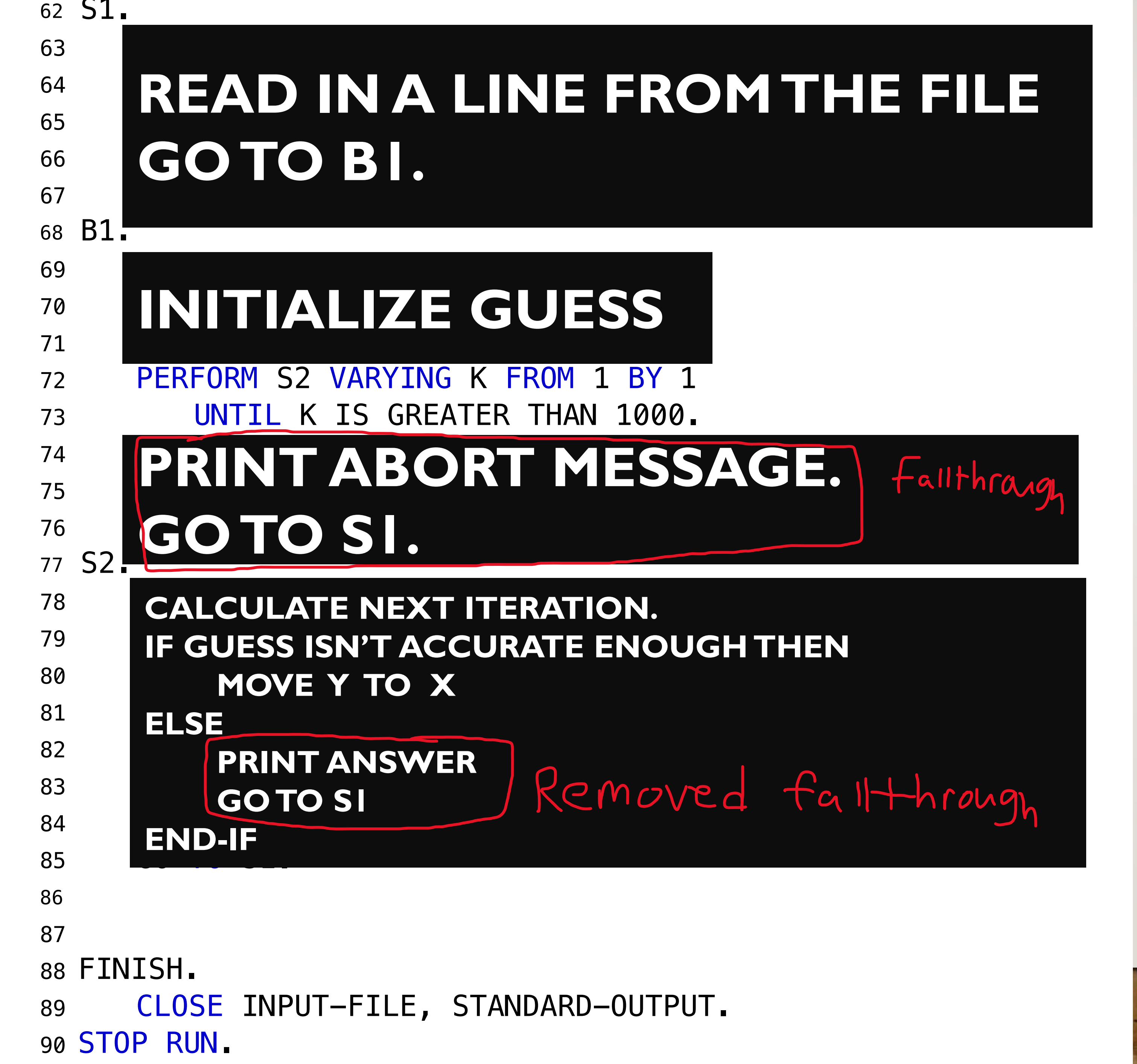
You could do

```
PERFORM
   DISPLAY “HELLO”
END-PERFORM
```



```
S1.
    READ IN A LINE FROM THE FILE
    GO TO B1.
B1.
    INITIALIZE GUESS
    PERFORM S2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO S1.
S2.
    CALCULATE NEXT ITERATION.
    IF GUESS ISN'T ACCURATE ENOUGH THEN
        MOVE Y TO X
    ELSE
        PRINT ANSWER
        GO TO S1
    END-IF


FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Then, using the same technique as before of removing fallthrough ambiguity, remove the **PRINT ABORT MESSAGE** fallthrough using a similar **IF-ELSE-ENDIF** structure.



```
S1.
    READ IN A LINE FROM THE FILE
    GO TO BI.
B1.
    INITIALIZE GUESS
    PERFORM S2 VARYING K FROM 1 BY 1
        UNTIL K IS GREATER THAN 1000.
    PRINT ABORT MESSAGE.
    GO TO SI.
S2.
    CALCULATE NEXT ITERATION.
    IF GUESS ISN'T ACCURATE ENOUGH THEN
          MOVE Y TO X
    ELSE
          PRINT ANSWER
          GO TO SI
    END-IF



FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

Fallthrough

Removed fallthrough

Then you could integrate **BI** into **SI**.

Your goal at that point would be to remove the **GO TO SI** constructs and replace the whole structure with a **while** loop (by that I mean **PERFORM until file_end = 1**)

Check this example by the prof.

At the end of the day though, file input will be replaced by user input, so this entire PowerPoint is a red herring.