SQRT.COB

Jason Nguyen

THE FILE

- Make sure you download the fixed SQRT.COB file from the A3 discussion
- Most of the code is constant string data used for print statements
- If you can trace programs with ease, you'll be fine
- I suggest taking several breaks between reading sessions, perhaps even over the course of several days

Analyzing SQRT.COB

All of it.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SQRT.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INPUT-FILE ASSIGN TO "sqrtFIXED.dat"
    ORGANIZATION IS LINE SEQUENTIAL.
    SELECT STANDARD-OUTPUT ASSIGN TO DISPLAY.
DATA DIVISION.
FILE SECTION.
FD INPUT-FILE.
    01 STANDARD-INPUT PICTURE X(80).
FD STANDARD-OUTPUT.
    01 OUT-LINE  PICTURE X(80).
WORKING-STORAGE SECTION.
77 DIFF PICTURE V9(5).
77 Z    PICTURE 9(11)V9(6).
77 K    PICTURE S9999.
77 X    PICTURE 9(11)V9(6).
77 Y    PICTURE 9(11)V9(6).
77 TEMP PICTURE 9(11)V9(6).
01 IN-CARD.
   02 IN-Z      PICTURE S9(10)V9(6) SIGN LEADING SEPARATE.
   02 IN-DIFF   PICTURE V9(5).
   02 FILLER    PICTURE X(58).
01 TITLE-LINE.
   02 FILLER PICTURE X(9) VALUE SPACES.
```

Useless header information

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SQRT.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INPUT-FILE ASSIGN TO "sqrtFIXED.dat"
    ORGANIZATION IS LINE SEQUENTIAL.
    SELECT STANDARD-OUTPUT ASSIGN TO DISPLAY.
DATA DIVISION.
FILE SECTION.
FD INPUT-FILE.
    01 STANDARD-INPUT PICTURE X(80).
FD STANDARD-OUTPUT.
    01 OUT-LINE  PICTURE X(80).
WORKING-STORAGE SECTION.
77 DIFF PICTURE V9(5).
77 Z    PICTURE 9(11)V9(6).
77 K    PICTURE S9999.
77 X    PICTURE 9(11)V9(6).
77 Y    PICTURE 9(11)V9(6).
77 TEMP PICTURE 9(11)V9(6).
01 IN-CARD.
   02 IN-Z      PICTURE S9(10)V9(6) SIGN LEADING SEPARATE.
   02 IN-DIFF   PICTURE V9(5).
   02 FILLER    PICTURE X(58).
01 TITLE-LINE.
   02 FILLER PICTURE X(9) VALUE SPACES.
```

- Recognize filename sqrtFIXED.dat
- Treat the file as line-by-line
- Assign stdout to display

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SQRT.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INPUT-FILE ASSIGN TO "sqrtFIXED.dat"
    ORGANIZATION IS LINE SEQUENTIAL.
    SELECT STANDARD-OUTPUT ASSIGN TO DISPLAY.
DATA DIVISION.
FILE SECTION.
FD INPUT-FILE.
    01 STANDARD-INPUT PICTURE X(80).
FD STANDARD-OUTPUT.
    01 OUT-LINE  PICTURE X(80).
WORKING-STORAGE SECTION.
77 DIFF PICTURE V9(5).
77 Z    PICTURE 9(11)V9(6).
77 K    PICTURE S9999.
77 X    PICTURE 9(11)V9(6).
77 Y    PICTURE 9(11)V9(6).
77 TEMP PICTURE 9(11)V9(6).
01 IN-CARD.
   02 IN-Z      PICTURE S9(10)V9(6) SIGN LEADING SEPARATE.
   02 IN-DIFF   PICTURE V9(5).
   02 FILLER    PICTURE X(58).
01 TITLE-LINE.
   02 FILLER PICTURE X(9) VALUE SPACES.
```

- INPUT-FILE is a file descriptor record that defines a standard input line as 80 characters
- STANDARD-OUTPUT is a file descriptor that uses OUT-LINE to print to the screen. It can print 80 characters too.

```
FILE SECTION.
FD INPUT-FILE.
    01 STANDARD-INPUT PICTURE X(80).
FD STANDARD-OUTPUT.
    01 OUT-LINE  PICTURE X(80).
WORKING-STORAGE SECTION.
77 DIFF PICTURE V9(5).
77 Z    PICTURE 9(11)V9(6).
77 K    PICTURE S9999.
77 X    PICTURE 9(11)V9(6).
77 Y    PICTURE 9(11)V9(6).
77 TEMP PICTURE 9(11)V9(6).
01 IN-CARD.
   02 IN-Z     PICTURE S9(10)V9(6) SIGN LEADING SEPARATE.
   02 IN-DIFF  PICTURE V9(5).
   02 FILLER   PICTURE X(58).
01 TITLE-LINE.
   02 FILLER PICTURE X(9) VALUE SPACES.
   02 FILLER PICTURE X(26) VALUE 'SQUARE ROOT APPROXIMATIONS'.
01 UNDER-LINE.
   02 FILLER PICTURE X(44) VALUE
      '--------------------------------------------'.
01 COL-HEADS.
   02 FILLER PICTURE X(8) VALUE SPACES.
   02 FILLER PICTURE X(6) VALUE 'NUMBER'.
   02 FILLER PICTURE X(15) VALUE SPACES.
   02 FILLER PICTURE X(11) VALUE 'SQUARE ROOT'.
```

Our variables.

```
FILE SECTION.
FD INPUT-FILE.
    01 STANDARD-INPUT PICTURE X(80).
FD STANDARD-OUTPUT.
    01 OUT-LINE  PICTURE X(80).
WORKING-STORAGE SECTION.
77 DIFF PICTURE V9(5).
77 Z    PICTURE 9(11)V9(6).
77 K    PICTURE S9999.
77 X    PICTURE 9(11)V9(6).
77 Y    PICTURE 9(11)V9(6).
77 TEMP PICTURE 9(11)V9(6).
01 IN-CARD.
   02 IN-Z     PICTURE S9(10)V9(6) SIGN LEADING SEPARATE.
   02 IN-DIFF  PICTURE V9(5).
   02 FILLER   PICTURE X(58).
01 TITLE-LINE.
   02 FILLER PICTURE X(9) VALUE SPACES.
   02 FILLER PICTURE X(26) VALUE 'SQUARE ROOT APPROXIMATIONS'.
01 UNDER-LINE.
   02 FILLER PICTURE X(44) VALUE
      '--------------------------------------------'.
01 COL-HEADS.
   02 FILLER PICTURE X(8) VALUE SPACES.
   02 FILLER PICTURE X(6) VALUE 'NUMBER'.
   02 FILLER PICTURE X(15) VALUE SPACES.
   02 FILLER PICTURE X(11) VALUE 'SQUARE ROOT'.
```

You don't have to memorize these yet.

This represents the last five numbers of each file-line. It is the epsilon, or the precision needed.

```
FILE SECTION.
FD INPUT-FILE.
    01 STANDARD-INPUT PICTURE X(80).
FD STANDARD-OUTPUT.
    01 OUT-LINE  PICTURE X(80).
WORKING-STORAGE SECTION.
77 DIFF PICTURE V9(5).
77 Z    PICTURE 9(11)V9(6).
77 K    PICTURE S9999.
77 X    PICTURE 9(11)V9(6).
77 Y    PICTURE 9(11)V9(6).
77 TEMP PICTURE 9(11)V9(6).
01 IN-CARD.
   02 IN-Z     PICTURE S9(10)V9(6) SIGN LEADING SEPARATE.
   02 IN-DIFF  PICTURE V9(5).
   02 FILLER   PICTURE X(58).
01 TITLE-LINE.
   02 FILLER PICTURE X(9) VALUE SPACES.
   02 FILLER PICTURE X(26) VALUE 'SQUARE ROOT APPROXIMATIONS'.
01 UNDER-LINE.
   02 FILLER PICTURE X(44) VALUE
      '--------------------------------------------'.
01 COL-HEADS.
   02 FILLER PICTURE X(8) VALUE SPACES.
   02 FILLER PICTURE X(6) VALUE 'NUMBER'.
   02 FILLER PICTURE X(15) VALUE SPACES.
   02 FILLER PICTURE X(11) VALUE 'SQUARE ROOT'.
```

You don't have to memorize these yet.

This is the input number. The radicand. The user input. You know.

```
FILE SECTION.
FD INPUT-FILE.
    01 STANDARD-INPUT PICTURE X(80).
FD STANDARD-OUTPUT.
    01 OUT-LINE  PICTURE X(80).
WORKING-STORAGE SECTION.
77 DIFF PICTURE V9(5).
77 Z    PICTURE 9(11)V9(6).
77 K    PICTURE S9999.
77 X    PICTURE 9(11)V9(6).
77 Y    PICTURE 9(11)V9(6).
77 TEMP PICTURE 9(11)V9(6).
01 IN-CARD.
   02 IN-Z     PICTURE S9(10)V9(6) SIGN LEADING SEPARATE.
   02 IN-DIFF  PICTURE V9(5).
   02 FILLER   PICTURE X(58).
01 TITLE-LINE.
   02 FILLER PICTURE X(9) VALUE SPACES.
   02 FILLER PICTURE X(26) VALUE 'SQUARE ROOT APPROXIMATIONS'.
01 UNDER-LINE.
   02 FILLER PICTURE X(44) VALUE
      '--------------------------------------------'.
01 COL-HEADS.
   02 FILLER PICTURE X(8) VALUE SPACES.
   02 FILLER PICTURE X(6) VALUE 'NUMBER'.
   02 FILLER PICTURE X(15) VALUE SPACES.
   02 FILLER PICTURE X(11) VALUE 'SQUARE ROOT'.
```

You don't have to memorize these yet.

This is just a variable used in the perform loop that counts to 1000.

```
FILE SECTION.
FD INPUT-FILE.
    01 STANDARD-INPUT PICTURE X(80).
FD STANDARD-OUTPUT.
    01 OUT-LINE  PICTURE X(80).
WORKING-STORAGE SECTION.
77 DIFF PICTURE V9(5).
77 Z    PICTURE 9(11)V9(6).
77 K    PICTURE S9999.
77 X    PICTURE 9(11)V9(6).
77 Y    PICTURE 9(11)V9(6).
77 TEMP PICTURE 9(11)V9(6).
01 IN-CARD.
   02 IN-Z     PICTURE S9(10)V9(6) SIGN LEADING SEPARATE.
   02 IN-DIFF  PICTURE V9(5).
   02 FILLER   PICTURE X(58).
01 TITLE-LINE.
   02 FILLER PICTURE X(9) VALUE SPACES.
   02 FILLER PICTURE X(26) VALUE 'SQUARE ROOT APPROXIMATIONS'.
01 UNDER-LINE.
   02 FILLER PICTURE X(44) VALUE
      '--------------------------------------------'.
01 COL-HEADS.
   02 FILLER PICTURE X(8) VALUE SPACES.
   02 FILLER PICTURE X(6) VALUE 'NUMBER'.
   02 FILLER PICTURE X(15) VALUE SPACES.
   02 FILLER PICTURE X(11) VALUE 'SQUARE ROOT'.
```

You don't have to memorize these yet.

These represent the previous guess and the current guess, respectively. Or, R_0 and R_1.

```
FILE SECTION.
FD INPUT-FILE.
    01 STANDARD-INPUT PICTURE X(80).
FD STANDARD-OUTPUT.
    01 OUT-LINE  PICTURE X(80).
WORKING-STORAGE SECTION.
77 DIFF PICTURE V9(5).
77 Z    PICTURE 9(11)V9(6).
77 K    PICTURE S9999.
77 X    PICTURE 9(11)V9(6).
77 Y    PICTURE 9(11)V9(6).
77 TEMP PICTURE 9(11)V9(6).
01 IN-CARD.
   02 IN-Z     PICTURE S9(10)V9(6) SIGN LEADING SEPARATE.
   02 IN-DIFF  PICTURE V9(5).
   02 FILLER   PICTURE X(58).
01 TITLE-LINE.
   02 FILLER PICTURE X(9) VALUE SPACES.
   02 FILLER PICTURE X(26) VALUE 'SQUARE ROOT APPROXIMATIONS'.
01 UNDER-LINE.
   02 FILLER PICTURE X(44) VALUE
      '--------------------------------------------'.
01 COL-HEADS.
   02 FILLER PICTURE X(8) VALUE SPACES.
   02 FILLER PICTURE X(6) VALUE 'NUMBER'.
   02 FILLER PICTURE X(15) VALUE SPACES.
   02 FILLER PICTURE X(11) VALUE 'SQUARE ROOT'.
```

You don't have to memorize these yet.

This is used to temporarily store the difference between the two guesses for the precision check.

```
01 IN-CARD.
   02 IN-Z      PICTURE S9(10)V9(6) SIGN LEADING SEPARATE.
   02 IN-DIFF   PICTURE V9(5).
   02 FILLER    PICTURE X(58).
01 TITLE-LINE.
   02 FILLER PICTURE X(9) VALUE SPACES.
   02 FILLER PICTURE X(26) VALUE 'SQUARE ROOT APPROXIMATIONS'.
01 UNDER-LINE.
   02 FILLER PICTURE X(44) VALUE
      '--------------------------------------------'.
01 COL-HEADS.
   02 FILLER PICTURE X(8) VALUE SPACES.
   02 FILLER PICTURE X(6) VALUE 'NUMBER'.
   02 FILLER PICTURE X(15) VALUE SPACES.
   02 FILLER PICTURE X(11) VALUE 'SQUARE ROOT'.
01 UNDERLINE-2.
   02 FILLER PICTURE X(20) VALUE ' ------------------'.
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 FILLER PICTURE X(19) VALUE '------------------'.
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
```

Each line read in from STANDARD-INPUT will be read into IN-CARD, which consists of:

- **IN-Z** – input number consisting of the sign (**S**), 10 digits (**9(10)**), an implied decimal point (**V**), and 6 digits after the decimal (**9(6)**). This temporary variable is moved to **Z**.
- **IN-DIFF** – input difference consisting of the five digits that come after the **IN-Z** data. It starts off with a decimal point (**V**) and then is followed by 5 digits after it.
- The **FILLER** is identical to struct padding in C. It's just to fill the rest of the line. There are 58 spaces after the **IN-Z** and **IN-DIFF**.

```
01 IN-CARD.
   02 IN-Z      PICTURE S9(10)V9(6) SIGN LEADING SEPARATE.
   02 IN-DIFF   PICTURE V9(5).
   02 FILLER    PICTURE X(58).
01 TITLE-LINE.
   02 FILLER PICTURE X(9) VALUE SPACES.
   02 FILLER PICTURE X(26) VALUE 'SQUARE ROOT APPROXIMATIONS'.
01 UNDER-LINE.
   02 FILLER PICTURE X(44) VALUE
      '--------------------------------------------'.
01 COL-HEADS.
   02 FILLER PICTURE X(8) VALUE SPACES.
   02 FILLER PICTURE X(6) VALUE 'NUMBER'.
   02 FILLER PICTURE X(15) VALUE SPACES.
   02 FILLER PICTURE X(11) VALUE 'SQUARE ROOT'.
01 UNDERLINE-2.
   02 FILLER PICTURE X(20) VALUE ' ------------------'.
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 FILLER PICTURE X(19) VALUE '------------------'.
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
```

These are all constant fluff. They are used to draw the table. That is all.

- **TITLE-LINE** is literally just spaces and the title.
- **UNDER-LINE** is literally the horizontal table line.
- **COL-HEADS** refers to the column headers.
- **UNDERLINE-2** is gay

```
01 IN-CARD.
   02 IN-Z      PICTURE S9(10)V9(6) SIGN LEADING SEPARATE.
   02 IN-DIFF   PICTURE V9(5).
   02 FILLER    PICTURE X(58).
01 TITLE-LINE.
   02 FILLER PICTURE X(9) VALUE SPACES.
   02 FILLER PICTURE X(26) VALUE 'SQUARE ROOT APPROXIMATIONS'.
01 UNDER-LINE.
   02 FILLER PICTURE X(44) VALUE
      '--------------------------------------------'.
01 COL-HEADS.
   02 FILL
   02 FILL
   02 FILL
   02 FILL
01 UNDERLI
   02 FILL
   02 FILL
   02 FILL
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
```

```
          Square Root Approximations
 ---------------------------------------------
          Number                 Square Root
 --------------------     ------------------
        5.000000                   2.236068
       -5.000000          Invalid Input
    91847.000000  Attempt aborted.Too many iterations
```

```
01 IN-CARD.
   02 IN-Z      PICTURE S9(10)V9(6) SIGN LEADING SEPARATE.
   02 IN-DIFF   PICTURE V9(5).
   02 FILLER    PICTURE X(58).
01 TITLE-LINE.
   02 FILLER PICTURE X(9) VALUE SPACES.
   02 FILLER PICTURE X(26) VALUE 'SQUARE ROOT APPROXIMATIONS'.
01 UNDER-LINE.
   02 FILLER PICTURE X(44) VALUE
      '--------------------------------------------'.
01 COL-HEADS.
   02 FILL
   02 FILL
   02 FILL
   02 FILL
01 UNDERLI
   02 FILL
   02 FILL
   02 FILL
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
```

```
        Square Root Approximations
 ---------------------------------------------
          Number                 Square Root
  --------------------     ------------------
               5.000000                2.236068
              -5.000000        Invalid Input
           91847.000000  Attempt aborted.Too many iterations
```

```
01 IN-CARD.
   02 IN-Z      PICTURE S9(10)V9(6) SIGN LEADING SEPARATE.
   02 IN-DIFF   PICTURE V9(5).
   02 FILLER    PICTURE X(58).
01 TITLE-LINE.
   02 FILLER PICTURE X(9) VALUE SPACES.
   02 FILLER PICTURE X(26) VALUE 'SQUARE ROOT APPROXIMATIONS'.
01 UNDER-LINE.
   02 FILLER PICTURE X(44) VALUE
      '--------------------------------------------'.
01 COL-HEADS.
   02 FILL
   02 FILL
   02 FILL
   02 FILL
01 UNDERLI
   02 FILL
   02 FILL
   02 FILL
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
```

```
          Square Root Approximations
--------------------------------------------
        Number                  Square Root
 --------------------     ------------------
              5.000000                2.236068
             -5.000000        Invalid Input
          91847.000000  Attempt aborted.Too many iterations
```

```
01 IN-CARD.
   02 IN-Z
   02 IN-DIFF
   02 FILLER
01 TITLE-LINE.
   02 FILLER P
   02 FILLER P
01 UNDER-LINE.
   02 FILLER PICTURE X(44) VALUE
      '--------------------------------------------'.
01 COL-HEADS.
   02 FILLER PICTURE X(8) VALUE SPACES.
   02 FILLER PICTURE X(6) VALUE 'NUMBER'.
   02 FILLER PICTURE X(15) VALUE SPACES.
   02 FILLER PICTURE X(11) VALUE 'SQUARE ROOT'.
01 UNDERLINE-2.
   02 FILLER PICTURE X(20) VALUE '  ------------------'.
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 FILLER PICTURE X(19) VALUE '------------------'.
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
```

```
          Square Root Approximations
 ---------------------------------------------
          Number                Square Root
  ------------------     ------------------
            5.000000                2.236068
           -5.000000        Invalid Input
        91847.000000  Attempt aborted.Too many iterations
```

```
01 IN-CARD.
   02 IN-Z
   02 IN-DIFF
   02 FILLER
01 TITLE-LINE.
   02 FILLER P
   02 FILLER P
01 UNDER-LINE.
   02 FILLER PICTURE X(44) VALUE
     '--------------------------------------------'.
01 COL-HEADS.
   02 FILLER PICTURE X(8) VALUE SPACES.
   02 FILLER PICTURE X(6) VALUE 'NUMBER'.
   02 FILLER PICTURE X(15) VALUE SPACES.
   02 FILLER PICTURE X(11) VALUE 'SQUARE ROOT'.
01 UNDERLINE-2.
   02 FILLER PICTURE X(20) VALUE ' -------------------'.
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 FILLER PICTURE X(19) VALUE '-------------------'.
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE -(11)9.9(6).
```

```
         Square Root Approximations
--------------------------------------------
        Number              Square Root
  -------------------   -------------------
          5.000000                 2.236068
         -5.000000          Invalid Input
      91847.000000  Attempt aborted.Too many iterations
```

```
01 IN-CARD.
   02 IN-Z
   02 IN-DIFF
   02 FILLER
01 TITLE-LINE.
   02 FILLER P
   02 FILLER P
01 UNDER-LINE.
   02 FILLER PICTURE X(44) VALUE
      '--------------------------------------------'.
01 COL-HEADS.
   02 FILLER PICTURE X(8) VALUE SPACES.
   02 FILLER PICTURE X(6) VALUE 'NUMBER'.
   02 FILLER PICTURE X(15) VALUE SPACES.
   02 FILLER PICTURE X(11) VALUE 'SQUARE ROOT'.
01 UNDERLINE-2.
   02 FILLER PICTURE X(20) VALUE '  ------------------'.
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 FILLER PICTURE X(19) VALUE '------------------'.
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
```

```
          Square Root Approximations
-------------------------------------------
        Number                Square Root
 ------------------     ------------------
              5.000000              2.236068
             -5.000000      Invalid Input
          91847.000000  Attempt aborted.Too many iterations
```

```
01 IN-CARD.
   02 IN-Z
   02 IN-DIFF
   02 FILLER
01 TITLE-LINE.
   02 FILLER P
   02 FILLER P
01 UNDER-LINE.
   02 FILLER PICTURE X(44) VALUE
      '--------------------------------------------'.
01 COL-HEADS.
   02 FILLER PICTURE X(8) VALUE SPACES.
   02 FILLER PICTURE X(6) VALUE 'NUMBER'.
   02 FILLER PICTURE X(15) VALUE SPACES.
   02 FILLER PICTURE X(11) VALUE 'SQUARE ROOT'.
01 UNDERLINE-2.
   02 FILLER PICTURE X(20) VALUE '  ------------------'.
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 FILLER PICTURE X(19) VALUE '------------------'.
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
```

```
          Square Root Approximations
-------------------------------------------------
        Number                  Square Root
 -------------------     -------------------
              5.000000                2.236068
             -5.000000        Invalid Input
          91847.000000  Attempt aborted.Too many iterations
```

```
01 IN-CARD.
   02 IN-Z
   02 IN-DIFF
   02 FILLER
01 TITLE-LINE.
   02 FILLER P
   02 FILLER P
01 UNDER-LINE.
   02 FILLER PICTURE X(44) VALUE
      '--------------------------------------------'.
01 COL-HEADS.
   02 FILLER PICTURE X(8) VALUE SPACES.
   02 FILLER PICTURE X(6) VALUE 'NUMBER'.
   02 FILLER PICTURE X(15) VALUE SPACES.
   02 FILLER PICTURE X(11) VALUE 'SQUARE ROOT'.
01 UNDERLINE-2.
   02 FILLER PICTURE X(20) VALUE ' --------------------'.
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 FILLER PICTURE X(19) VALUE '--------------------'.
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
```

```
          Square Root Approximations
--------------------------------------------
        Number                 Square Root
 --------------------     -------------------
              5.000000                2.236068
             -5.000000         Invalid Input
          91847.000000  Attempt aborted.Too many iterations
```

```
01 IN-CARD.
   02 IN-Z
   02 IN-DIFF
   02 FILLER
01 TITLE-LINE.
   02 FILLER P
   02 FILLER P
01 UNDER-LINE.
   02 FILLER PICTURE X(44) VALUE
      '--------------------------------------------'.
01 COL-HEADS.
   02 FILLER PICTURE X(8) VALUE SPACES.
   02 FILLER PICTURE X(6) VALUE 'NUMBER'.
   02 FILLER PICTURE X(15) VALUE SPACES.
   02 FILLER PICTURE X(11) VALUE 'SQUARE ROOT'.
01 UNDERLINE-2.
   02 FILLER PICTURE X(20) VALUE ' --------------------'.
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 FILLER PICTURE X(19) VALUE '-------------------'.
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
```

```
          Square Root Approximations
-----------------------------------------------
        Number                  Square Root
 ------------------      --------------------
              5.000000                2.236068
             -5.000000        Invalid Input
          91847.000000  Attempt aborted.Too many iterations
```

```
01 IN-CARD.
   02 IN-Z
   02 IN-DIFF
   02 FILLER
01 TITLE-LINE.
   02 FILLER P
   02 FILLER P
01 UNDER-LINE.
   02 FILLER PICTURE X(44) VALUE
      '--------------------------------------------'.
01 COL-HEADS.
   02 FILLER PICTURE X(8) VALUE SPACES.
   02 FILLER PICTURE X(6) VALUE 'NUMBER'.
   02 FILLER PICTURE X(15) VALUE SPACES.
   02 FILLER PICTURE X(11) VALUE 'SQUARE ROOT'.
01 UNDERLINE-2.
   02 FILLER PICTURE X(20) VALUE '  ------------------'.
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 FILLER PICTURE X(19) VALUE '------------------'.
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
```

```
          Square Root Approximations
-------------------------------------------
        Number                Square Root
 -------------------     -------------------
              5.000000                2.236068
             -5.000000        Invalid Input
          91847.000000  Attempt aborted.Too many iterations
```

```
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
   02 FILLER PICTURE X(21) VALUE '         INVALID INPUT'.
01 ABORT-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUTP-Z PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(37) VALUE
      '  ATTEMPT ABORTED,TOO MANY ITERATIONS'.
```

PRINT-LINE is used for printing the table's next row upon successful calculation.

```
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
   02 FILLER PICTURE X(21) VALUE '        INVALID INPUT'.
01 ABORT-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUTP-Z PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(37) VALUE
      '  ATTEMPT ABORTED,TOO MANY ITERATIONS'.
```

```
          Square Root Approximations
-----------------------------------------------
         Number                  Square Root
 --------------------     -------------------
            5.000000                 2.236068
           -5.000000         Invalid Input
        91847.000000   Attempt aborted.Too many iterations
```

```
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
   02 FILLER PICTURE X(21) VALUE '        INVALID INPUT'.
01 ABORT-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUTP-Z PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(37) VALUE
      '  ATTEMPT ABORTED,TOO MANY ITERATIONS'.
```

The **Z** is used instead of **9** for when you want to suppress leading 0s. Otherwise if you use **9**, it will always show a 0.

```
          Square Root Approximations
-----------------------------------------
         Number              Square Root
 -------------------     ------------------
              5.000000              2.236068
             -5.000000       Invalid Input
          91847.000000  Attempt aborted.Too many iterations
```

```
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
   02 FILLER PICTURE X(21) VALUE '        INVALID INPUT'.
01 ABORT-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUTP-Z PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(37) VALUE
      '  ATTEMPT ABORTED,TOO MANY ITERATIONS'.
```

```
          Square Root Approximations
-----------------------------------------
        Number                Square Root
 -------------------     -------------------
           5.000000              2.236068
          -5.000000         Invalid Input
       91847.000000   Attempt aborted.Too many iterations
```

```
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
   02 FILLER PICTURE X(21) VALUE '         INVALID INPUT'.
01 ABORT-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUTP-Z PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(37) VALUE
      '  ATTEMPT ABORTED,TOO MANY ITERATIONS'.
```

```
          Square Root Approximations
-----------------------------------------------
         Number                 Square Root
 --------------------     -------------------
              5.000000                2.236068
             -5.000000          Invalid Input
          91847.000000  Attempt aborted.Too many iterations
```

```
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
   02 FILLER PICTURE X(21) VALUE '        INVALID INPUT'.
01 ABORT-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUTP-Z PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(37) VALUE
      '  ATTEMPT ABORTED,TOO MANY ITERATIONS'.
```

ERROR-MESS is used for printing the table's next row upon invalid input. It is effectively—as the name suggests—an error message.

```
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
   02 FILLER PICTURE X(21) VALUE '         INVALID INPUT'.
01 ABORT-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUTP-Z PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(37) VALUE
      '  ATTEMPT ABORTED,TOO MANY ITERATIONS'.
```

```
          Square Root Approximations
-----------------------------------------
        Number                Square Root
 -------------------     ------------------
              5.000000                2.236068
             -5.000000        Invalid Input
          91847.000000  Attempt aborted.Too many iterations
```

```
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
   02 FILLER PICTURE X(21) VALUE '         INVALID INPUT'.
01 ABORT-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUTP-Z PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(37) VALUE
      '  ATTEMPT ABORTED,TOO MANY ITERATIONS'.
```

```
          Square Root Approximations
-----------------------------------------------
         Number                 Square Root
 -------------------     -------------------
              5.000000                2.236068
             -5.000000        Invalid Input
          91847.000000  Attempt aborted.Too many iterations
```

```
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
   02 FILLER PICTURE X(21) VALUE '         INVALID INPUT'.
01 ABORT-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUTP-Z PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(37) VALUE
      '  ATTEMPT ABORTED,TOO MANY ITERATIONS'.
```

```
          Square Root Approximations
-----------------------------------------
         Number                Square Root
 -------------------     ------------------
              5.000000                2.236068
             -5.000000          Invalid Input
          91847.000000  Attempt aborted.Too many iterations
```

```
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
   02 FILLER PICTURE X(21) VALUE '         INVALID INPUT'.
01 ABORT-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUTP-Z PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(37) VALUE
      '  ATTEMPT ABORTED,TOO MANY ITERATIONS'.
```

ABORT-MESS is a rare error message where if the square root doesn't resolve after a number of iterations, we time out and move on.

```
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
   02 FILLER PICTURE X(21) VALUE '        INVALID INPUT'.
01 ABORT-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUTP-Z PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(37) VALUE
      '  ATTEMPT ABORTED,TOO MANY ITERATIONS'.
```

```
          Square Root Approximations
-----------------------------------------------
         Number                 Square Root
 -------------------     -------------------
             5.000000                2.236068
            -5.000000         Invalid Input
         91847.000000  Attempt aborted.Too many iterations
```

```
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
   02 FILLER PICTURE X(21) VALUE '         INVALID INPUT'.
01 ABORT-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUTP-Z PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(37) VALUE
      '  ATTEMPT ABORTED,TOO MANY ITERATIONS'.
```

```
          Square Root Approximations
-----------------------------------------------
          Number                 Square Root
 --------------------     --------------------
              5.000000                2.236068
             -5.000000         Invalid Input
          91847.000000   Attempt aborted.Too many iterations
```

```
01 PRINT-LINE.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUT-Z  PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(5) VALUE SPACES.
   02 OUT-Y  PICTURE Z(11)9.9(6).
01 ERROR-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OT-Z   PICTURE -(11)9.9(6).
   02 FILLER PICTURE X(21) VALUE '         INVALID INPUT'.
01 ABORT-MESS.
   02 FILLER PICTURE X VALUE SPACE.
   02 OUTP-Z PICTURE Z(11)9.9(6).
   02 FILLER PICTURE X(37) VALUE
      '  ATTEMPT ABORTED,TOO MANY ITERATIONS'.
```

```
          Square Root Approximations
-----------------------------------------
         Number                Square Root
 -------------------     -------------------
              5.000000                2.236068
             -5.000000         Invalid Input
          91847.000000   Attempt aborted.Too many iterations
```

TAKE A SMALL BREAK.

WE HAVEN'T EVEN BEGUN.

```
PROCEDURE DIVISION.
    OPEN INPUT INPUT-FILE, OUTPUT STANDARD-OUTPUT.
    WRITE OUT-LINE FROM TITLE-LINE AFTER ADVANCING 0 LINES.
    WRITE OUT-LINE FROM UNDER-LINE AFTER ADVANCING 1 LINE.
    WRITE OUT-LINE FROM COL-HEADS AFTER ADVANCING 1 LINE.
    WRITE OUT-LINE FROM UNDERLINE-2 AFTER ADVANCING 1 LINE.
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.
```

The **PROCEDURE DIVISION** is where all the actual magic happens. Think of it like **main()** in procedural languages like C/Java.

```
PROCEDURE DIVISION.
    OPEN INPUT INPUT-FILE, OUTPUT STANDARD-OUTPUT.
    WRITE OUT-LINE FROM TITLE-LINE AFTER ADVANCING 0 LINES.
    WRITE OUT-LINE FROM UNDER-LINE AFTER ADVANCING 1 LINE.
    WRITE OUT-LINE FROM COL-HEADS AFTER ADVANCING 1 LINE.
    WRITE OUT-LINE FROM UNDERLINE-2 AFTER ADVANCING 1 LINE.
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.
```

These are two separate statements that open the **INPUT-FILE** and open the **STANDARD-OUTPUT** (printing out to screen).

```
PROCEDURE DIVISION.
    OPEN INPUT INPUT-FILE, OUTPUT STANDARD-OUTPUT.
    WRITE OUT-LINE FROM TITLE-LINE AFTER ADVANCING 0 LINES.
    WRITE OUT-LINE FROM UNDER-LINE AFTER ADVANCING 1 LINE.
    WRITE OUT-LINE FROM COL-HEADS AFTER ADVANCING 1 LINE.
    WRITE OUT-LINE FROM UNDERLINE-2 AFTER ADVANCING 1 LINE.
S1.
```

```
          Square Root Approximations
-----------------------------------------
        Number                 Square Root
 ------------------     ------------------
             5.000000                2.236068
            -5.000000        Invalid Input
         91847.000000  Attempt aborted.Too many iterations
```

```
        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.
```

These four statements just print the four constant table headers/borders we talked about earlier.

The program consists of printing these four lines, then for every line it takes from the SQRT.DAT, it prints out one of the three different results lines mentioned earlier (**OUT-LINE, ERROR-MESS, ABORT-MESS**).

AFTER ADVANCING _ LINES is just how you specify the number of newlines to print after the statement. I personally prefer using **DISPLAY**.

```
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.
```

This is the first paragraph that runs: **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.
```

It consists of these instructions.

```
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.
```

First, it attempts to read a line from the file into the **IN-CARD** struct mentioned earlier.

```
01 IN-CARD.
   02 IN-Z      PICTURE S9(10)V9(6) SIGN
LEADING SEPARATE.
   02 IN-DIFF   PICTURE V9(5).
   02 FILLER    PICTURE X(58).
```

```
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.
```

After reading it in, it checks if the inputted number is positive. Only positive numbers can be square-rooted.

```
01 IN-CARD.
   02 IN-Z      PICTURE S9(10)V9(6) SIGN
LEADING SEPARATE.
   02 IN-DIFF   PICTURE V9(5).
   02 FILLER    PICTURE X(58).
```

```
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.
```

Let's look at the shorter-lasting outcome. If the input number **IN-Z** is negative… it doesn't **go to B1** and just goes forward.

```
01 IN-CARD.
   02 IN-Z      PICTURE S9(10)V9(6) SIGN
LEADING SEPARATE.
   02 IN-DIFF   PICTURE V9(5).
   02 FILLER    PICTURE X(58).
```

```
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
    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 means the condition just falls through. It's more confusing than **if-else** hence why we want to remove **go to**.

```
                Square Root Approximations
-----------------------------------------------------
          Number                  Square Root
 --------------------      ------------------
                5.000000                  2.236068
               -5.000000          Invalid Input
```

```
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.
```

We go back to the start of the paragraph, **S1**.

I think it is evident that this paragraph just serves to read in a new line from the file repeatedly until there are no longer lines to give (as you can see by the **AT END GO TO FINISH**).

You may wish to change the name of **S1** for readability and ease of future refactoring.

```
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.
```

We go back to the start of the paragraph, **S1**.

I think it is evident that this paragraph just serves to read in a new line from the file repeatedly until there are no longer lines to give (as you can see by the **AT END GO TO FINISH**).

You may wish to change the name of **S1** for readability and ease of future refactoring.

```
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.
```

We go back to the start of the paragraph, **S1**.

I think it is evident that this paragraph just serves to read in a new line from the file repeatedly until there are no longer lines to give (as you can see by the **AT END GO TO FINISH**).

You may wish to change the name of **S1** for readability and ease of future refactoring.

```
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.
```

We go back to the start of the paragraph, **S1**.

I think it is evident that this paragraph just serves to read in a new line from the file repeatedly until there are no longer lines to give (as you can see by the **AT END GO TO FINISH**).

You may wish to change the name of **S1** for readability and ease of future refactoring.

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO GO TO B1.
```

Okay, so we closed that fork of the maze. Let's see the what would happen if our number were positive (=> square rootable).

```
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.
```

```
01 IN-CARD.
   02 IN-Z      PICTURE S9(10)V9(6) SIGN
LEADING SEPARATE.
   02 IN-DIFF   PICTURE V9(5).
   02 FILLER    PICTURE X(58).
```

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.
```

Okay, so we closed that fork of the maze. Let's see the what would happen if our number were positive (=> square rootable).

```
01 IN-CARD.
   02 IN-Z      PICTURE S9(10)V9(6) SIGN
LEADING SEPARATE.
   02 IN-DIFF   PICTURE V9(5).
   02 FILLER    PICTURE X(58).
```

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.
```

If the number can be square rooted, we **go to B1**.

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.
```

This moves the input number and the input epsilon into our temporary work variables, **DIFF** and **Z**.

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.
```

Remember that **X** is our first (or 'previous') guess. By default, the program's first guess, **X**, is half of the original number, **Z**.

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.
```

This **PERFORM** statement is probably one of the most difficult aspects of the project. This is analogous to a **for** loop in C. What this **PERFORM** loop is saying is:

PERFORM S2 then E2, and every time you do these two paragraphs, add 1 to K. This repeats 1000 times.

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.
```

At first, you might think that it is doing the Babylon algorithm 1000 times for accuracy. But in reality, it is the timeout check. Why?

This is another example of annoying fallthrough from earlier. **The loop is not supposed to finish!** If it finishes, it falls through to the statements below it, which is the **"I did this 1000 times and I'm calling it quits because that's too many**" message.

Unfortunately, due to the way the code is styled (how the loop isn't supposed to finish and reach the next instruction), this means that somewhere in the **S2** and **E2** paragraph, there must be another **go to S1** instruction that sends us back to the beginning… see why **go to**s are hated?

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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
    IF TEMP / (Y + X) IS GREATER THAN
    MOVE IN-Z TO OUT-Z.
    MOVE Y TO OUT-Y.
    WRITE OUT-LINE FROM PRINT-LINE AF
    GO TO S1.
E2.
    MOVE Y TO X.
FINISH.
    CLOSE INPUT-FILE, STANDARD-OUTPUT.
STOP RUN.
```

But, just like last time with the **ERROR-MESS** fork in the maze, the **ABORT-MESS** fork appears to be a short path as well, so we should get rid of it while we can.

```
 Square Root Approximations
-------------------------------------------
        Number                Square Root
 -------------------     ------------------
              5.000000                2.236068
             -5.000000        Invalid Input
          91847.000000  Attempt aborted.Too many iterations
```

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.
```

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

SIGH...

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

We've already checked out the fork of the maze where we sent an invalid input so let's just stay positive and **go to B1**.

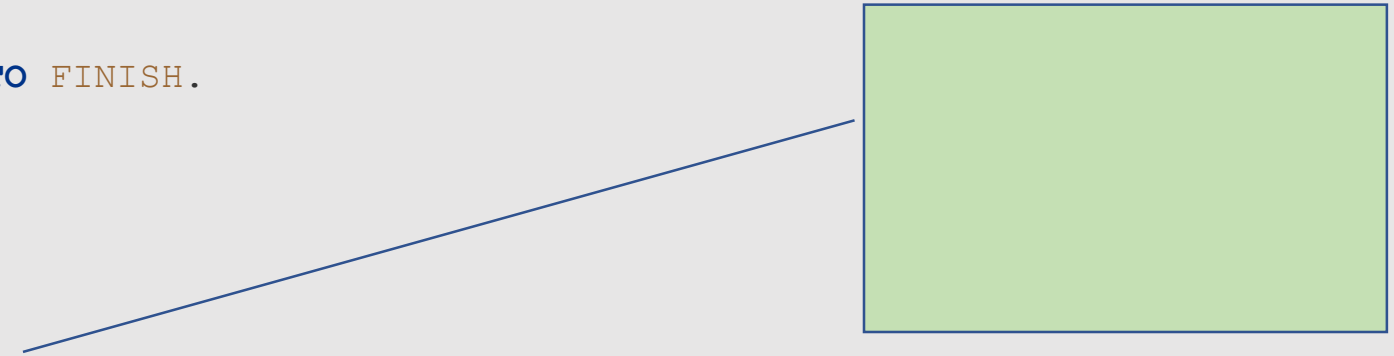
```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO GO TO B1.



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.



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.
```



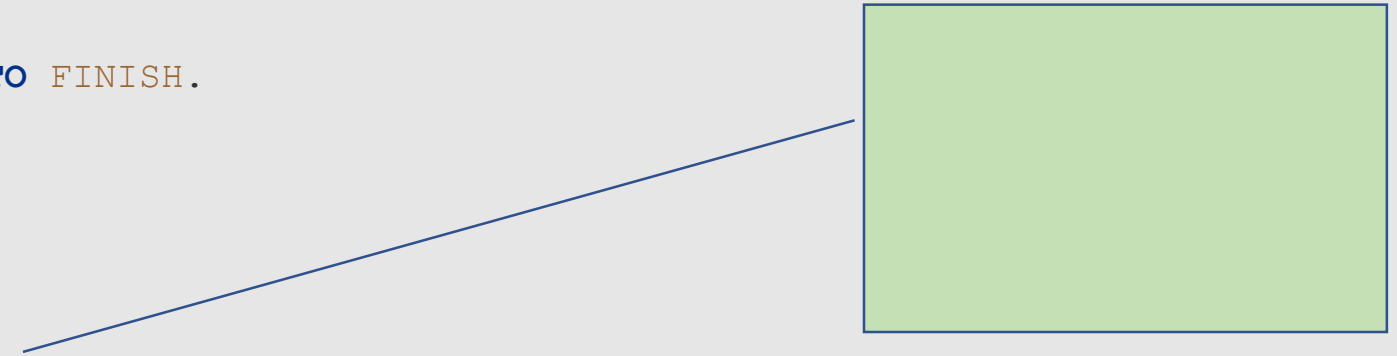
```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO GO TO B1.



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.



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.
```



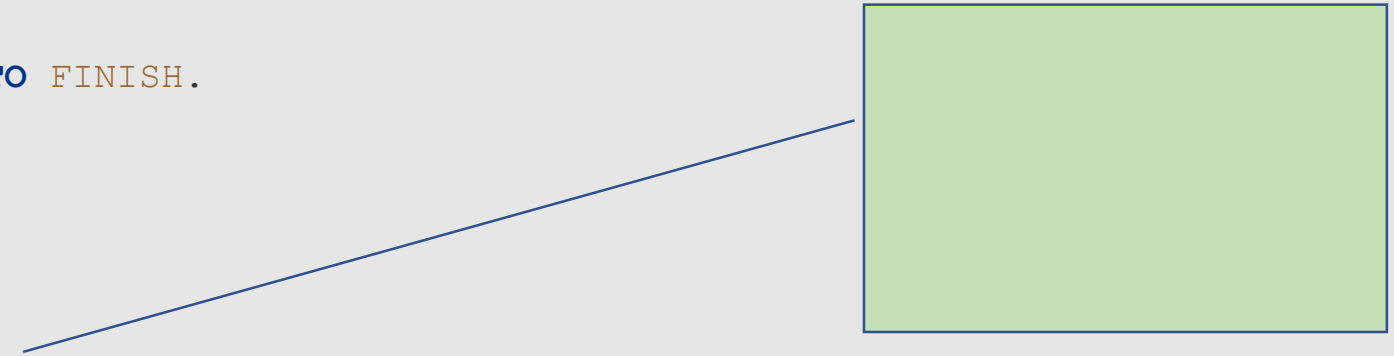
```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO GO TO B1.



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.



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 INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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 think it's important to establish what is going on from a big picture point of view.

K is currently **1**

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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 think it's important to establish what is going on from a big picture point of view.

K is currently **1**

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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 think it's important to establish what is going on from a big picture point of view.

K is currently **1**

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.
```

I think it's important to establish what is going on from a big picture point of view.

K is currently **1**

```
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 INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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 think it's important to establish what is going on from a big picture point of view.

K is currently **1**

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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 think it's important to establish what is going on from a big picture point of view.

K is currently **1**

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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 think it's important to establish what is going on from a big picture point of view.

K is currently **2**

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.
```

I think it's important to establish what is going on from a big picture point of view.

K is currently **2**

```
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 INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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 think it's important to establish what is going on from a big picture point of view.

K is currently **3**

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.
```

I think it's important to establish what is going on from a big picture point of view.

K is currently **3**

```
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 INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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 think it's important to establish what is going on from a big picture point of view.

K is currently **4**

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

OK you get the point. Let's actually dive in.

K is currently **9999999999**

(you might want to rename **B1** now if you have an idea of what it is. I don't view it as the actual **Babylon**, but as a driver function for it, or a **setup**).

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

OK you get the point.
Let's actually dive in.

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

Y is the next guess. It is calculated from the previous guess, **X**. It looks weird in this legacy COBOL program, but as you can see from the assignment, it is:

$$R_1 = \frac{(R_0 + \frac{N}{R_0})}{2}$$

This was explained in the previous PPT:

nextGuess =
(prevGuess + (number / prevGuess)) / 2

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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 from the previous presentation that we only keep two variables for the guesses so that we can compare them, so now we are comparing them. We are finding the **absolute error** between the two guesses by subtracting them.

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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 is a poor man's **abs()** function. If the number is negative, make it positive.

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

Here is the precision check. Recall that **DIFF** (our epsilon) refers to the **last five digits** of each line, representing the desired precision. Most of the lines in the **SQRT.DAT** file end in **00100**, so using that as an example, we are checking the following inequality:

TEMP / (Y + X) > 0.001, or:

abs(Y – X) / (Y + X) > 0.001

You don't have to worry much about the meaning of this. All that matters is that this is a measure of error between the two. This algorithm converges logarithmically and can close in on an accurate answer very quickly. **As we close in on the right answer, the differences between subsequent answers becomes smaller and smaller**. Here, 0.001 (or any other **DIFF**) is our goal difference!! Our **epsilon**!! In proper math terms.

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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 **if** statement only executes its consequence—**go to E2**—when the error measurement is **greater than** the goal. When the two guesses **X** and **Y** are too different, their error measurement will be greater than our target **DIFF**/epsilon. This means **E2** is our “do it again” paragraph.

This is where things get annoying btw.

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

What this does is it moves the most recent guess, **Y**, to the previous guess variable, **X**. Then nothing happens.

…or so you may think. Recall what happens after **E2**.

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

Hahahahaha welcome back to hell. The developer of this program relied on the fallback nature of this **PERFORM** loop in order for it to work. So we see **S2** as the paragraph that **actually** does a single **Babylon** calculation, and **E2** is the paragraph that re-feeds the guess back into **S2**. We can now think of the **PERFORM** loop as follows:

```
for (int K = 1; K <= 1000; K++) {
      result = calculateBabylon(guess);
      guess = result;
}
```

This is what the developer intended, but look at this insidious abuse of explicit transfer of control. Disgusting. ;)

This is one of the most important realizations of this project. Don't bother continuing if you don't understand this.

lmao

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

Hahahahaha welcome back to hell. The developer of this program relied on the fallback nature of this **PERFORM** loop in order for it to work. So we see **S2** as the paragraph that **actually** does a single **Babylon** calculation, and **E2** is the paragraph that re-feeds the guess back into **S2**. We can now think of the **PERFORM** loop as follows:

```
for (int K = 1; K <= 1000; K++) {
      result = calculateBabylon(guess);
      guess = result;
}
```

This is what the developer intended, but look at this insidious abuse of explicit transfer of control. Disgusting. ;)

March 17 update: this loop is much harder to explain than I thought. If you truly don't understand at this point, go back to the doc and read "**Understanding the Most Difficult GO TO in the Assignment**"

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

So, now that we got that out of the way (it took me hours to figure that out), we can look at the alternate path.

Going back, we are still checking on the relative error of the two guesses, **X** and **Y**. We figured out that if this was greater than our epsilon / **DIFF**, then it “wasn’t good enough” and we sent it to **E2** to re-feed the guess back into the algorithm.

Now, here is our third fallthrough. This fallthrough is actually not an error fallthrough as we had to deal with the last two times. This is actually our success line.

```
S1.
```

```
 Square Root Approximations
-------------------------------------------
        Number                 Square Root
 ------------------     ------------------
            5.000000                2.236068
           -5.000000        Invalid Input
```

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



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.
```

The first instruction moves the input number into the output number (which is just used for print formatting—don't worry).

The second instruction does the same thing, but instead of the input number, it moves the last guess into the output guess variable.

Finally, it prints out the **OUT-LINE** (success line) onto the table.

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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 expected, at the end of the **ABORT-MESS**, we **go to S1**; and at the end of the **ERROR-MESS**, we **go to S1**…and…at the end of printing a successful line, we **go to S1** to receive our next line. If you haven't renamed your **S1** at this point, you are doing yourself a major disservice.

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

So the overwhelming majority of inputs will go through the following main path:

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

So the overwhelming majority of inputs will go through the following main path:

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.

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.
```

So the overwhelming majority of inputs will go through the following main path:

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.

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.
```

So the overwhelming majority of inputs will go through the following main path:

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.

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.
```

So the overwhelming majority of inputs will go through the following main path:

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.

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.
```

So the overwhelming majority of inputs will go through the following main path:

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.
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.
```

So the overwhelming majority of inputs will go through the following main path:

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.

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.
```

So the overwhelming majority of inputs will go through the following main path:

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.

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.
```

So the overwhelming majority of inputs will go through the following main path:

(don't forget we increase K by 1 every loop)

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.

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.
```

And this continuously repeats until we reach the desired accuracy from the epsilon…

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.

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.
```

And this continuously repeats until we reach the desired accuracy from the epsilon…

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.

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.
```

Until . . . eventually . . .

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.

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.
```

Until . . . eventually . . .

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.

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.
```

Until . . . eventually . . .

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.

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.
```

Until . . . eventually . . .

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.

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.
```

Until . . . eventually . . .

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

. . . it finishes! Now, that's one loop. Let's finish off the bigger loop (the one created from **go to**) . . . .

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

. . . it finishes! Now, that's one loop. Let's finish off the bigger loop (the one created from **go to**) . . . .

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

. . . it finishes! Now, that's one loop. Let's finish off the bigger loop (the one created from **go to**) . . . .

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

. . . it finishes! Now, that's one loop. Let's finish off the bigger loop (the one created from **go to**) . . . .

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

. . . it finishes! Now, that's one loop. Let's finish off the bigger loop (the one created from **go to**) . . . .

Just going to move slide this to the right for some artistic merit…

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.

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.
```

. . . it finishes! Now, that's one loop. Let's finish off the bigger loop (the one created from **go to**) . . . .

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.

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.
```

. . . it finishes! Now, that's one loop. Let's finish off the bigger loop (the one created from **go to**) . . . .

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

Until...

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

Until...

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

Until...

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

Until...

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

Until...

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

Until...

```
S1.
    READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
    IF IN-Z IS GREATER THAN ZERO 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.



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.
```

The program terminates!!! Freedom!!

WI

- He

```
56 PROCEDURE DIVISION.
57     OPEN INPUT INPUT-FILE, OUTPUT STANDARD-OUTPUT.
58     WRITE OUT-LINE FROM TITLE-LINE AFTER ADVANCING 0 LINES.
59     WRITE OUT-LINE FROM UNDER-LINE AFTER ADVANCING 1 LINE.
60     WRITE OUT-LINE FROM COL-HEADS AFTER ADVANCING 1 LINE.
61     WRITE OUT-LINE FROM UNDERLINE-2 AFTER ADVANCING 1 LINE.
62 S1.
63     READ INPUT-FILE INTO IN-CARD AT END GO TO FINISH.
64     IF IN-Z IS GREATER THAN ZERO GO TO B1.
65     MOVE IN-Z TO OT-Z.
66     WRITE OUT-LINE FROM ERROR-MESS AFTER ADVANCING 1 LINE.
67     GO TO S1.
68 B1.
69     MOVE IN-DIFF TO DIFF.
70     MOVE IN-Z TO Z.
71     DIVIDE 2 INTO Z GIVING X ROUNDED.
72     PERFORM S2 THRU E2 VARYING K FROM 1 BY 1
73         UNTIL K IS GREATER THAN 1000.
74     MOVE IN-Z TO OUTP-Z.
75     WRITE OUT-LINE FROM ABORT-MESS AFTER ADVANCING 1 LINE.
76     GO TO S1.
77 S2.
78     COMPUTE Y ROUNDED = 0.5 * (X + Z / X).
79     SUBTRACT X FROM Y GIVING TEMP.
80     IF TEMP IS LESS THAN ZERO COMPUTE TEMP = - TEMP.
81     IF TEMP / (Y + X) IS GREATER THAN DIFF GO TO E2.
82     MOVE IN-Z TO OUT-Z.
83     MOVE Y TO OUT-Y.
84     WRITE OUT-LINE FROM PRINT-LINE AFTER ADVANCING 1 LINE.
85     GO TO S1.
86 E2.
87     MOVE Y TO X.
88 FINISH.
89     CLOSE INPUT-FILE, STANDARD-OUTPUT.
90 STOP RUN.
```

error if input is -ve

accuracy not reached error

output calculated root

FLOWC

- Here is a fl