Consider a set of simultaneous linear equations:

\[
\begin{align*}
A_{1,1}x_1 + A_{1,2}x_2 + A_{1,3}x_3 + \ldots + A_{1,N}x_N &= A_{1,N+1} \\
A_{2,1}x_1 + A_{2,2}x_2 + A_{2,3}x_3 + \ldots + A_{2,N}x_N &= A_{2,N+1} \\
A_{3,1}x_1 + A_{3,2}x_2 + A_{3,3}x_3 + \ldots + A_{3,N}x_N &= A_{3,N+1} \\
&\quad \vdots \\
A_{N,1}x_1 + A_{N,2}x_2 + A_{N,3}x_3 + \ldots + A_{N,N}x_N &= A_{N,N+1}
\end{align*}
\]

where \(x_i\) are variables and \(A_{i,j}\) are constants, most of which are coefficients of the variables (though the so-called “non-homogeneous terms”, arranged to appear on the right-hand side of each equation, don’t multiply any variables). The first index \((i)\) of each coefficient identifies the equation or row the coefficient is in, and the second index \((j)\) identifies the variable that the constant multiplies, which is also the column in which that variable is located because the variables are arranged in columns. There are \(N\) variables and \(N\) equations. There are \(N+1\) constants in each equation organized in \(N+1\) columns, for a total of \(N \times (N+1)\) constants. Hence, \(i\) varies from 1 to \(N\) and \(j\) varies from 1 to \(N+1\).

The Gaussian elimination algorithm uses the equations to eliminate terms in the other equations, without changing the location of each variable in each equation, until they look like this and the solution is clear:

\[
\begin{align*}
1 \cdot x_1 + 0 \cdot x_2 + 0 \cdot x_3 + \ldots + 0 \cdot x_N &= A_{1,N+1} \\
0 \cdot x_1 + 1 \cdot x_2 + 0 \cdot x_3 + \ldots + 0 \cdot x_N &= A_{2,N+1} \\
0 \cdot x_1 + 0 \cdot x_2 + 1 \cdot x_3 + \ldots + 0 \cdot x_N &= A_{3,N+1} \\
&\quad \vdots \\
0 \cdot x_1 + 0 \cdot x_2 + 0 \cdot x_3 + \ldots + 1 \cdot x_N &= A_{N,N+1}
\end{align*}
\]

(Note that the values of the constants on the right-hand side generally change as a result of this process).
In the Gaussian elimination algorithm we focus on the constant coefficients because they are the only “players” that change in any way. The variables \( x_i \) don’t change location or otherwise play much more of a role than simply placeholders. [Note that this isn’t true if we use the more popular “elimination by substitution” procedure, in which the terms keep moving around and require more work to keep track of.] Moreover, the “+” operators and the “=” signs also play no truly active role, so for visual clarity it helps to drop them from the notation, understanding that they are still really there. The equations can therefore be represented solely in terms of the coefficients, as long as we don’t move them around:

\[
\begin{array}{cccc}
A_{1,1} & A_{1,2} & A_{1,3} & A_{1,N} & A_{1,N+1} \\
A_{2,1} & A_{2,2} & A_{2,3} & A_{2,N} & A_{2,N+1} \\
A_{3,1} & A_{3,2} & A_{3,3} & A_{3,N} & A_{3,N+1} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
A_{N,1} & A_{N,2} & A_{N,3} & A_{N,N} & A_{N,N+1} \\
\end{array}
\]

The coefficients above in bold face lie on what we call the “main diagonal” of the set of \( N \times N \) coefficients on the left-hand side of the “=” sign. The main diagonal terms have something in common—on each, the row index equals the column index. We can write them in the general form \( A_{k,k} \), where \( k \) can have any value from 1 to \( N \).

The Gaussian elimination strategy uses these coefficients to eliminate and modify the other coefficients until they look like this:

\[
\begin{array}{cccc}
1 & 0 & 0 & 0 & A_{1,N+1} \\
0 & 1 & 0 & 0 & A_{2,N+1} \\
0 & 0 & 1 & 0 & A_{3,N+1} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 1 & A_{N,N+1} \\
\end{array}
\]

(Again, the values of the terms in the right-most column change as a result of this process and, when all is done, they represent the solution.)
The Gaussian elimination strategy goes like this:

1. Multiply (a copy of) the first row by \(-A_{2,1}/A_{1,1}\) and add the results, term by term, to the second row. This will eliminate the first term of the second row (that is, make it 0). Then, multiply (a copy of) the first row by \(-A_{3,1}/A_{1,1}\) and add the results, term by term, to the third row to eliminate the first term in the third row. Repeat this procedure to eliminate the first term of each of the subsequent equations.

Here’s a generalized way of saying this that will help us convert the procedure into pseudocode: Use the 1st term in the 1st row to eliminate the first term in the \(i\)th row (where \(i = 2, \ldots, N\)) by multiplying the first row by \(-A_{i,1}/A_{1,1}\) and adding the results, term by term, to the \(i\)th row.

2. For this to work, the coefficient \(A_{1,1}\) must not be zero. Hence, before we carry out step 1 we should check to see if \(A_{1,1}\) is zero. If it is zero, we should search the other rows, one by one, until we find one whose first term is not zero, and exchange that equation (term by term) with the first equation. If we can’t find any equations with a non-zero first term, then there are, in effect, no equations with \(x_1\) in them. Hence, we would have \(N\) equations but only \(N-1\) unknowns, which would over-constrain the variables.

(What does “over-constrain” mean? Well, each equation imposes a “constraint” on the variables—that is, it limits the values that the variables can have and still satisfy the equation. Each equation imposes an additional constraint, and if you impose \(N\) constraints on \(N\) variables, then there will be only one solution that satisfies them all. If you impose \(N+1\) constraints on \(N\) variables, there won’t be any way for the variables to satisfy all the equations—the variables are over-constrained—and there won’t be any solution. If this is the case, then we should issue an error message and stop the solution procedure.)

3. Repeat the two steps above, this time using the 2nd term of the 2nd row to eliminate the 2nd term from each of the rows below it. (First make sure that the 2nd row has a non-zero second term in it, swapping rows if necessary. If none can be found, an error message must be issued and the procedure stopped.) Repeat this using the 3rd term in the 3rd row to eliminate the 3rd term in the subsequent equations, and so on, through the \((N-1)\)st row, eliminating the \((N-1)\)st term in the \(N\)th equation.
To generalize the three previous steps:

Check to make sure that the \( k \)th coefficient in the \( k \)th row (a main diagonal coefficient) is not zero. If it is zero, search rows below until a non-zero coefficient in the \( k \)th column is found and swap the \( k \)th row with that row term-by-term. If no such coefficient can be found, issue an error message (the equations are probably over-constrained) and stop.

Multiply (a copy of) the \( k \)th row by \(-A_{i,k}/A_{k,k}\) and add the results term-by-term to the \( i \)th row, to eliminate the \( k \)th term in the \( i \)th row.

We’ve named indices so that \( k \) represents the row whose main diagonal coefficient, \( A_{k,k} \), is used to eliminate the \( k \)th coefficient (that is, the coefficient in the \( k \)th column) from each of the rows below it. Hence, in steps (1) and (2) \( k \) varies from 1 to \( N-1 \). The index \( i \) represents the rows in which the \( k \)th coefficient is being eliminated. Hence, in this process \( i \) varies from \( k+1 \) to \( N \).

When we’re done we should have something that looks like this:

\[
\begin{pmatrix}
A_{1,1} & A_{1,2} & A_{1,3} & A_{1,N} & A_{1,N+1} \\
0 & A_{2,2} & A_{2,3} & A_{2,N} & A_{2,N+1} \\
0 & 0 & A_{3,3} & A_{3,N} & A_{3,N+1} \\
& & & & \\
0 & 0 & 0 & A_{N,N} & A_{N,N+1}
\end{pmatrix}
\]

(Note that during this process, the values of all coefficients below the 1st row might have have been modified.)

In the \( N \)th equation we’ve eliminated all variables except \( x_N \), so the \( N \)th equation now reads: \( A_{N,N} \cdot x_N = A_{N,N+1} \). To solve for \( x_N \), we divide the \( N \)th equation by \( A_{N,N} \), but first we need to check that \( A_{N,N} \) is not zero.

Check to see if \( A_{N,N} = 0 \). If so, then the \( N \)th equation says that \( 0 = A_{N,N+1} \). If \( A_{N,N+1} = 0 \), then any value of \( x_N \) will satisfy the equation—that is, there are an infinite number of solutions to the equation \( 0 \cdot x_N = 0 \), so print a message to that effect and stop. However, if \( A_{N,N+1} \neq 0 \), then no value of \( x_N \) will satisfy the equation \( 0 \cdot x_N = A_{N,N+1} \neq 0 \), and so there are no solutions to the equations as a whole. Print an appropriate message and stop.

However, if \( A_{N,N} \neq 0 \), then solve for \( x_N \) by dividing the \( N \)th equation by \( A_{N,N} \).
Now the coefficients look like this:

\[
\begin{array}{cccccc}
A_{1,1} & A_{1,2} & A_{1,3} & A_{1,N} & A_{1,N+1} \\
0 & A_{2,2} & A_{2,3} & A_{2,N} & A_{2,N+1} \\
0 & 0 & A_{3,3} & A_{3,N} & A_{3,N+1} \\
& & & & & \cdots \\
0 & 0 & 0 & 1 & A_{N,N+1}
\end{array}
\]

(where the value of \(A_{N,N+1}\) has changed after step (3) above).

We’re not done yet, of course—we still have to eliminate the coefficients above the main diagonal and solve for each variable. But that is simpler in part because we now know that all of the main diagonal terms are not zero, so we don’t have to check them first. Hence, lets further refine the three steps outlined above.

---

*** Partial pseudocode for Gaussian elimination algorithm. ***

[Open input and output files and read the value of \(N\) and the values of the \(N \times N+1\) constant coefficient values from the input file.]

*** Loop to eliminate terms below the main diagonal. Index \(k\) refers to the row whose main diagonal coefficient is used to eliminate coefficients in the \(k\)th column below it. ***

For \(k\) from 1 to \(N-1\), Do

*** Starting with the main diagonal coefficient in the current \((k)\)th row, find the first non-zero coefficient in the \(k\)th column below it. Index \(i\) keeps track of the row being checked for such a non-zero coefficient. ***

\[
i \leftarrow k
\]

While \(A_{i,k} = 0\) and \(i \leq N\), Do

\[
i \leftarrow i + 1
\]

End While
*** Determine the consequences of the search.***

If \( i = N + 1 \), then

*** We weren’t able to find a non-zero coefficient in the \( k \)th column on or below the main diagonal. This means that the equations are over- or possibly under-constrained. ***

Print message
Stop

Else If \( i > k \), then

*** We found a non-zero coefficient, but it was not on the main diagonal \((i > k)\). Hence, swap the coefficient values in the \( i \)th row with those in the \( k \)th row. (Since the coefficients in columns 1 through \( k-1 \) in both rows have already been eliminated, there’s no need to swap those—they’re all zero—so we start swapping terms in the \( k \)th column.) Note that to swap terms between two rows in any particular column, we must first store the value of one of the terms temporarily (“Hold”) so that it doesn’t get overwritten before we swap its value into the other row. ***

Do for \( j \) from \( k \) to \( N+1 \)
   Hold \( \leftarrow A_{k,j} \)
   \( A_{k,j} \leftarrow A_{i,j} \)
   \( A_{i,j} \leftarrow \) Hold
End Do

End If

*** Multiply (a copy of) terms in the \( k \)th row by \(-A_{i,k}/A_{k,k}\) and add the result term by term to terms in the \( i \)th row, where \( i \) is a row somewhere from the row just below the \( k \)th row \((k+1)\) down to the \( N \)th row. Since the first \( k-1 \) coefficients in the \( k \)th row are now zero, don’t bother to include them in this process—they won’t contribute anything. Moreover, since we know that \( A_{i,k} \) will become zero as a result of this process, we don’t need to bother actually to compute the change—we can simply set it to zero if we want to. [Really, though, since we won’t actually use this value again, there’s no need even to do that, except perhaps for temporary debugging or confidence-building purposes.] ***
Do for $i$ from $k+1$ to $N$

*** Temporarily store (in “Hold”) the factor used to eliminate $A_{i,k}$ (beneath the main diagonal term, $A_{k,k}$). This factor is the same for all terms in the current row, so there’s no point in recomputing it for each column in the row. ***

$\text{Hold} \leftarrow -A_{i,k}/A_{k,k}$

Do for $j$ from $k+1$ to $N+1$

$A_{i,j} \leftarrow A_{i,j} + \text{Hold} \cdot A_{k,j}$

End Do  (End of $j$ loop through columns modified in $i$th row)

***As a temporary debugging or confidence-building step, set $A_{i,k}$ to zero, which was the whole point of the preceding step. However, this coefficient isn’t used again, so there’s no actual need to set it to zero or anything else. ***

*** $[A_{i,k} \leftarrow 0]$***

End Do  (End of $i$ loop through rows being modified)

End Do  (End of $k$ loop through rows used to modify other rows)

*** Check $A_{N,N}$ to see whether a solution exists and is unique. ***

If $A_{N,N} = 0$, then

*** Either there are no solutions or an infinite number of them. Figure out which. ***

If $A_{N,N+1} = 0$, then

*** There are an infinite number of solutions. ***

Print message.

Else

*** There are no solutions.***
Print message.

End If

Stop

End If

*** For each individual variable, $x_k$, solve for it and immediately use it to eliminate coefficients above the main diagonal (that is, in the $k$th column in rows from $k$-1 to 1). The solved-for variables used to eliminate their brethren lie on the main diagonal in rows N upward through row 2. When done, solve for the remaining variable, $x_1$. [Note that when solving for $x_k$, we already know that $A_{k,k}$ will become 1, so there is need to carry out the division of $A_{k,k}$ by itself. In fact, $A_{k,k}$ won’t actually be used any more at all, so there is no need to change it any further, even simply to set it to 1! Insert code to change it to 1, but only for temporary debugging and confidence-building purposes. ***

[Try writing this part of the code on your own, looking for opportunities to make the code more efficient by not making unnecessary computations and excluding unnecessary assignment statements, except temporarily for debugging or confidence-building purposes. Hint: the rest of the code is relatively simple compared to the foregoing—only seven more lines of code, plus whatever is needed to write the output. (If you’re really clever and understand well enough how loops work, then you can reduce this to six lines of code by making the $k$-index of the outer loop run from N-1 to 1 instead of N-1 to 2.) And don’t forget that you can make Do loops count backwards if you want, simply by specifying a negative increment!]