There are multiple ways to read the binary numbers from `std::cin`. One (rather tedious) way is to read the input character-wise. This solution most likely requires two nested loops which can quickly become hard to read for a third person.

Another way to read the 8-bit binary numbers from the input is by using the following trick: we can simply treat the numbers as if they were decimal numbers and thus read each number into a variable of type `int`. Notice that this works without loss, since the largest (positive) number a 32-bit `int` can represent is 2147483647, thus it can store more than 8 decimal digits. All that then remains to do is to extract the true value of the binary number from the `int` and then to convert that value to `char`.

```cpp
1 // Informatik − Serie 8 − Aufgabe 1
2 // Programm: nzz_decoder.cpp
3 // Autor: ... (Gruppe ...)
4 // Reads until the end of file 8−bit binary numbers and outputs the
5 // characters they represent in ASCII.
6 //
7 //#include "tests.h"
8 #include <iostream>
9 #include <cassert>
10
11 // PRE: (i) the decimal digits in character are all either 0 or 1,
12 // (ii) character requires at most 8 decimal digits
13 // POST: returns the char corresponding to the embedded number
14 char binary_to_char (int character) {
15   assert(character / 100000000 == 0); // verify PRE-condition (ii)
16   // Transform embedded binary number to true value
17   int character_value = 0;
18   for (int digit_power = 1; digit_power <= 128; digit_power *= 2) {
19     const int digit = character % 10; // extract last embedded digit
20     assert(digit == 0 || digit == 1); // verify PRE-condition (i)
21     character_value += digit * digit_power;
22     character /= 10; // remove last embedded digit
23   } // (e.g. 00000011 --> 0000001)
24   return static_cast<char>(character_value);
25 }
```
Solution 2

The word \( w_0 \) is always the initial word \( s_0 \). For \( n > 0 \) we obtain the word \( w_n \) from \( w_{n-1} \) by substituting each symbol in \( w_{n-1} \) with its production.

a) \( w_0: \) F-
\( w_1: \) F+F-
\( w_2: \) F+F+F+F-

b) \( w_0: \) ac
\( w_1: \) bd
\( w_2: \) ca
\( w_3: \) db
\( w_4: \) ac

c) \( w_0: \) B
\( w_1: \) FFF
\( w_2: \) FBFBFB
\( w_3: \) FBFFFFBFFFFFBFFFF

Solution 3

First of all, notice that the last lines in each code piece simply output the given matrix. i.e. these lines:

```cpp
for (int i = 0; i < ...; ++i) {
    for (int j = 0; j < ...; ++j)
        std::cout << mat[i][j] << " ";
    std::cout << "\n";
}
```
Now about the actual solution:

a) Output:
```
T H E
E T H
```
Explanation: In the first line of the snippet we initialize the array as
```
a b c
d e f
```
In the next 6 lines we overwrite these values one by one. Pay attention to which element is overwritten in each step:
```
a H c
d e f
a H c
d e H
T H c
d e H
T H c
d T H
T H E
d T H
T H E
E T H
```
In the remaining lines we simply output the multidimensional array.

b) Output:
```
0 1 2
1 2 3
```
Explanation: We first start by initializing a 2 × 3-matrix with elements of type `int` and initial value 1. In lines 2 to 4 we fill the matrix row-wise. The first row is simply filled with the column-indices (i.e. `j`) because `i` is 0. The second row is filled with the column-indices shifted by 1 as here the row-index `i` is equal to 1. In the remaining lines we simply output the multidimensional vector.

c) Output:
```
1 0 0
0 1 0
0 0 1
```
Explanation: We first start with an uninitialized 3 × 3-matrix. In lines 2 to 8 we walk over the matrix row-wise as above, and we assign each element the value 1 if and only if it is on the diagonal (i.e. where `i == j`) and 0 otherwise. In the remaining lines we simply output the multidimensional array.
Solution 4

Note: the following programs could also be written such that they use functions, for example to initialize the matrices, perform the matrix multiplication, etc.

a)

```cpp
#include <iostream>

int main() {
    // Input
    int n;
    std::cin >> n;
    double d;
    std::cin >> d;

    // Build matrix
    std::vector<std::vector<double>> diag(n, std::vector<double>(n, 0));
    for (int i = 0; i < n; ++i)
        diag[i][i] = d;

    // Output matrix
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j)
            std::cout << diag[i][j] << " ";
        std::cout << "\n";
    }

    return 0;
}
```

b)

```cpp
#include <iostream>

int main() {
    // Input
    const int m = 3;
    const int n = 3;
    double A[m][n];
```
// Build C = A*B
double C[m][n];
for (int i = 0; i < m; ++i)
    for (int j = 0; j < n; ++j) {
        // Get entry C[i][j], i.e. the product-sum of row i of A
        // with column j of B
        C[i][j] = 0;
        for (int k = 0; k < n; ++k)
            C[i][j] += A[i][k] * B[k][j];
    }

// Output
for (int i = 0; i < m; ++i) {
    for (int j = 0; j < n; ++j)
        std::cout << C[i][j] << " ";
    std::cout << "\n";
}
for (int col = 1; col < n; ++col)
    for (int row = 0; row < m; ++row)
        V[row][col] = V[row][col-1] * x[row];

// Output
for (int row = 0; row < m; ++row) {
    for (int col = 0; col < n; ++col)
        std::cout << V[row][col] << " ";
    std::cout << "\n";
}

return 0;