C Programming, Exercises for the sixth week

1. (Compulsory) Write a function
termNode* fromByteToPolynomial(uint8_t byte);

that constructs a polynomial from a byte. The coefficients of the polynomial
are modulo 2 integers and the degree at most 7. For example, if the byte is
10011011, then the polynomial is $1 + X^3 + X^4 + X^6 + X^7$.

2. (Compulsory) Write a function
uint8_t fromPolynomialToByte(termNode* P);

that outputs a byte corresponding the polynomial $P$ which is at most of degree
7 and with modulo 2 coefficients.

3. (Compulsory) Write a function
void mixColumns(state* st);

that mixes the state array $st$ in the following way. Let

\[
\begin{pmatrix}
  c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\
  c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\
  c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\
  c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3}
\end{pmatrix}
\]

be the original state array and

\[
\begin{pmatrix}
  02 & 03 & 01 & 01 \\
  01 & 02 & 03 & 01 \\
  01 & 01 & 02 & 03 \\
  03 & 01 & 01 & 02
\end{pmatrix}
\]

a constant $4 \times 4$ matrix (in hexadecimal). Perform the matrix multiplication
of these two matrices, when elements are interpreted as polynomials. The addi-
tion of the polynomials is modulo 2 addition. The multiplication is first
done normally as a modulo 2 polynomial multiplication. If the degree of the
resulting polynomial is greater than 7, it is divided (modulo 2 division) by
$X^8 + X^4 + X^3 + X + 1$ and the remainder is taken as a result.

Remember the formula for the matrix multiplication:

\[
\begin{pmatrix}
  b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\
  b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\
  b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\
  b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3}
\end{pmatrix}
\times
\begin{pmatrix}
  c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} \\
  c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} \\
  c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} \\
  c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3}
\end{pmatrix}
\]

\[
= \begin{pmatrix}
  d_{0,0} & d_{0,1} & d_{0,2} & d_{0,3} \\
  d_{1,0} & d_{1,1} & d_{1,2} & d_{1,3} \\
  d_{2,0} & d_{2,1} & d_{2,2} & d_{2,3} \\
  d_{3,0} & d_{3,1} & d_{3,2} & d_{3,3}
\end{pmatrix}
\]
The elements $d_{ij}$ in the product matrix are calculated by the formula

$$d_{ij} = \sum_{k=0}^{3} b_{ik} c_{kj}.$$

MixColumns is the main confusing function in AES (Advanced Encryption Standard). In modern encryption, plaintext is confused many times before the actual encryption takes place. In AES, the functions shiftRows, subBytes and mixColumns are used to confuse the plaintext before the encryption is done.

4. Write the function

```c
void invMixColumns(state* st);
```

that does the same as mixColumns, but this time using the constant matrix (in hexadecimal notation)

```
0E 0B 0D 09
09 0E 0B 0D
0D 09 0E 0B
0B 0D 09 0E
```

5. Write a function void addRoundKey(state* st, state* key); which XORs st and key. The key should be a random bit string whose length is the same as the plaintext block (thus 128 bits).

6. Now you can perform one round AES encryption. Let the plaintext be already in a state array st. The encryption consists of the following steps:

```c
subBytes(st, sbox);
shiftRows(st);
mixColumns(st);
addRoundKey(st, key);
```

The decryption proceeds as follows:

```c
addRoundKey(st, key);
invMixColumns(st);
invShiftRows(st);
subBytes(st, invBox);
```

The subBytes in the decryption is the same as in the encryption, but this time it uses the inverse Sbox which is given below. In order the subBytes to work, the indices made by constructIndices must be right. Test this. For example, if the byte is 0100 0010, then the index $i$ is 0100 or 2 and the index $j$ is 0010 or 4.
Make the encryption and decryption and check if you get the original plaintext. If you can get your system to work, you will get 5 points from this single task.

Inverse Sbox:

52 09 6a d5 30 36 a5 38 bf 40 a3 9e 81 f3 d7 fb
7c e3 39 82 9b 2f ff 87 34 8e 43 44 c4 de e9 cb
54 7b 94 32 a6 c2 23 3d ee 4c 95 0b 42 fa c3 4e
08 2e a1 66 28 d9 24 b2 76 5b a2 49 6d 8b d1 25
72 f8 f6 64 86 68 98 16 d4 a4 5c cc 5d 65 b6 92
6c 70 48 50 fd ed b9 da 5e 15 46 57 a7 8d 9d 84
90 d8 ab 00 8c bc d3 0a f7 e4 58 05 b8 b3 45 06
d0 2c 1e 8f ca 3f 0f 02 c1 af bd 03 01 13 8a 6b
3a 91 11 41 4f 67 dc ea 97 f2 cf ce f0 b4 e6 73
96 ac 74 22 e7 ad 35 85 e2 f9 37 e8 1c 75 df 6e
47 f1 1a 71 1d 29 c5 89 6f b7 62 0e a8 1b be 1b
fc 56 3e 4b c6 d2 79 20 9a db c0 fe 78 cd 5a f4
1f dd a8 33 88 07 c7 31 b1 12 10 59 27 80 ec 5f
60 51 7f a9 19 b5 4a 0d 2d e5 7a 9f 93 c9 9c ef
a0 e0 3b 4d ae 2a f5 b0 c8 eb bb 3c 83 53 99 61
17 2b 04 7e ba 77 d6 26 e1 69 14 63 55 21 0c 7d

Observations. You can test the confusion modules shiftRows, subBytes and mixColumns separately. Apply for example shiftRows to a state array, and after this at once invShiftRows. You should get the original state array as a result. Do the same with subBytes(st, box) and subBytes(st, invbox), and mixColumns and invMixColumns. If you are going to show your encryption to work, prepare many test cases (plaintext blocks which you encrypt and decrypt).

This is only only round AES. The standard AES makes the encryption 16 times, every time with a different key generated from a secret master key.

A real encryption software must be fast. Because the polynomials in AES are modulo 2 polynomials, it is possible to use bitwise operations instead of general polynomial operations. For example, addition of polynomials can be done with a bitwise xor.

7. Write a function

```c
void shuffle(void* array, int len, size_t elem_size);
```

that shuffles the contents of arrays of any type. The parameter `elem_size` contains the size of a single element in bytes. Use the generic swap function presented in the lecture of the second week.

Example call:
double nums[5] = { 1.5, 3.5, 5.5, 7.5, 9.5 };  
shuffle(nums, 5, sizeof(double));  
print_doubles(nums, 5); // implemented in a previous task

Example result:

5.500000, 1.500000, 9.500000, 3.500000, 7.500000

8. Write a function maxi() which takes three parameters, two double values $x$ and $y$ and a double function $f(double)$, and returns the larger of the two values $f(x)$ and $f(y)$. Test your function with various parameter functions.