```c
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

#define MAX_WORD        4294967296
#define TAU0            0x61707865
#define TAU1            0x3320646e
#define TAU2            0x79622d32
#define TAU3            0x6b206574

typedef unsigned long uint_32;

uint_32 rotate(uint_32 a, uint_32 b) {
    uint_32 rotate;

    while(b != 0) {
        a *= 2;
        b--;
    }
    rotate = a / MAX_WORD;
    a %= MAX_WORD;
    a += rotate;
    a %= MAX_WORD;
    return a;
}

void quarterround(uint_32 *y0, uint_32 *y1, uint_32 *y2, uint_32 *y3) {
    *y1 ^= rotate((*y0 + *y3) % MAX_WORD, 7);
    *y2 ^= rotate((*y1 + *y0) % MAX_WORD, 9);
    *y3 ^= rotate((*y2 + *y1) % MAX_WORD, 13);
    *y0 ^= rotate((*y3 + *y2) % MAX_WORD, 18);
}

void columnround(uint_32 y[]) {
    quarterround(&y[0], &y[4], &y[8], &y[12]);
    quarterround(&y[5], &y[9], &y[13], &y[1]);
    quarterround(&y[10], &y[14], &y[2], &y[6]);
    quarterround(&y[15], &y[3], &y[7], &y[11]);
}

void rowround(uint_32 y[]) {
    quarterround(&y[0], &y[1], &y[2], &y[3]);
    quarterround(&y[5], &y[6], &y[7], &y[4]);
    quarterround(&y[10], &y[11], &y[8], &y[9]);
    quarterround(&y[15], &y[12], &y[13], &y[14]);
}

void doubleround(uint_32 y[]) {
    columnround(y);
    rowround(y);
}

void Salsa20(uint_32 x[], uint_32 r, uint_32 output[]){
    uint_32 xc[16];
    int i;
```

```
    memcpy(xc, x, sizeof(uint_32)*16);
    for(i = 0; i < r; i++)
        doubleround(xc);
    for(i = 0; i < 16; i++) {
        xc[i] = (x[i] + xc[i]) % MAX_WORD;
    }
    for(i = 0; i < 64; i += 4) {
        output[i] = xc[i/4] % 256;
        xc[i/4] -= output[i];
        xc[i/4] /= 256;
        output[i + 1] = xc[i/4] % 256;
        xc[i/4] -= output[i + 1];
        xc[i/4] /= 256;
        output[i + 2] = xc[i/4] % 256;
        xc[i/4] -= output[i + 2];
        xc[i/4] /= 256;
        output[i + 3] = xc[i/4] % 256;
        xc[i/4] -= output[i + 3];
        xc[i/4] /= 256;
    }
}

void initCounter(uint_32 *i) {
    i[0] = 0;
    i[1] = 0;
}

void addCounter(uint_32 *i0, uint_32 *i1) {
    if((*i0) < MAX_WORD)
        (*i0)++;
    else {
        (*i0) = 0;
        (*i1)++;
    }
}

char* encryptSalsa20(uint_32 *x, uint_32 r, char *m, uint_32 size) {
    int j;
    uint_32 output[64];
    char *cy = (char*)malloc(sizeof(char) * (size + 1));
    char *c = cy;
    while(size != 0) {
        Salsa20(x, 10, output);
        addCounter(&x[8], &x[9]);
        if(size <= 64) {
            for(j = 0; j < size; j++) {
                c[j] = m[j]^output[j];
            }
            c[j] = '\0';
            return cy;
        }
        else {
            for(j = 0; j < 64; j++) {
                c[j] = m[j]^output[j];
            }
            size -= 64;
            c += 64;
```

```
            m += 64;
        }
    }
}

char* decryptSalsa20(uint_32 *x, uint_32 r, char *m, uint_32 size) {
    x[8] = 0;
    x[9] = 0;
    return encryptSalsa20(x, r, m, size);
}

void setX(uint_32 *X, uint_32 *k, uint_32 *nonce, uint_32 *i){
    // constant diagonal
    X[0] = TAU0;
    X[5] = TAU1;
    X[10]= TAU2;
    X[15]= TAU3;
    // key
    X[1] = k[0];
    X[2] = k[1];
    X[3] = k[2];
    X[4] = k[3];
    X[11]= k[4];
    X[12]= k[5];
    X[13]= k[6];
    X[14]= k[7];
    // nonce
    X[6] = nonce[0];
    X[7] = nonce[1];
    // ith-block
    X[8] = i[0];
    X[9] = i[1];
}

int setKey(uint_32 *key) {
    char k[33];
    int i;
    int opcao;
    char arquivo[50];

    printf("Entre com uma chave de 32 caracteres:\n");
    fgets(k, 33, stdin);

    for(i = 0; i < 8; i++) {
        key[i] = k[4*i] + k[4*i + 1]*256 + k[4*i + 2]*65536 + k[4*i + 3]*16777216;
    }
    return EXIT_SUCCESS;
}

void setNonce(uint_32 *nonce) {
    srand(time(NULL));
    nonce[0] = rand() + rand() + 1;
    nonce[1] = rand() + rand() + 1;
}

char* readMessage() {
    int opcao;
```

```
    int pos = 0;
    int newSize = 1;
    char *message;

    message = (char*)malloc(sizeof(char) * 1000);

    printf("\n\nEntre com o texto a ser criptografado (Digite fim para terminar):\n");
    do {
       fgets(message + pos, 1000, stdin);
       pos = strlen(message);
       if(pos > 900 * newSize) {
              newSize++;
              message = (char*)realloc(message, sizeof(char)*newSize*1000);
       }
    } while(strcmp(&message[pos - 4], "fim\n"));
    message[pos - 4] = '\0';

    return message;
}

int main(int argc, char** argv) {
    uint_32 x[16];
    uint_32 key[8];
    uint_32 nonce[2];
    uint_32 i[2];
    char *message;
    char *cyphertext;
    char *recoveredMessage;
    int j, k;
    uint_32 size;

    setKey(key);
    setNonce(nonce);
    initCounter(i);
    setX(x, key, nonce, i);
    message = readMessage();
    if(message == NULL)
        return EXIT_FAILURE;
    size = strlen(message);
    cyphertext = encryptSalsa20(x, 10, message, size);
    free(message);
    printf("\n\nTexto Cifrado:\n");
    for(j = 1; j <= size; j++) {
          printf("0x%08x ", cyphertext[j-1]);
          if (j%7==0) printf("\n");
    }
    printf("\n");
    recoveredMessage = decryptSalsa20(x, 10, cyphertext, size);
    printf("\n\nMensagem Recuperada:\n");
    for(j = 1; j <= size; j++) {
          printf("         %c ", recoveredMessage[j-1]);
          if (j%7==0) printf("\n");
    }
    printf("\n");
    free(cyphertext);
    free(recoveredMessage);
    return 0;
```

```
}
```