[pipe] Crypto

Il programma *crypto* contenuto nell'archivio allegato invia, **in forma cifrata**, una serie di frasi di Einstein della forma:

```
ALBERT EINSTEIN: L'uomo ha scoperto la bomba atomica, pero' nessun topo al mondo costruirebbe una trappola per topi.#
```

- Ogni frase inizia con `"ALBERT EINSTEIN:"` e termina con `"#"`;
- La frase viene cifrata utilizzando una chiave random da 1 a 9 che viene sottratta al codice ASCII di ogni carattere. Ad esempio con la chiave 1 `ALBERT` diventa `@KADQS` (notare che `@` precede `A` nella tabella ASCII).

Il programma **si aspetta in input la frase decifrata** e solo a quel punto ne propone una nuova della stessa forma ma **cifrata con una nuova chiave**. Solo se tutte le frasi vengono decifrate correttamente il programma stampa:

```
[*] Congratulazioni! Hai superato il crypto quiz!
```

Normalmente il programma legge e scrive su terminale. E' possibile specificare due pipe come argomenti da riga di comando usando rispettivamente le opzioni `-i` (input) e `-o` (output). In tale caso crypto **crea** le due pipe e le utilizza per comunicare. Alla fine dell'esecuzione le pipe vengono **eliminate**. Ad esempio:

```
./crypto -i /tmp/pipeIn -o /tmp/pipeOut
```

Il programma supporta altre opzioni:

- `-v` attiva la modalità verbosa.
- `-h` mostra le opzioni supportate.

Segue un esempio di esecuzione dove l'interazione avviene tramite standard input e standard output. Le frasi che iniziano con ALBERT EINSTEIN sono state inserite da terminale. Solo la prima è corretta e dopo la seconda si nota un messaggio di errore (per replicare questo test usare l'opzione `-v` che mostra la frase in chiaro)

```
$ ./crypto
>I?BOQBFKPQBFK7I$rj^kfq^$^so^$i^ploqb`ebp^mo^$jbofq^opf+
ALBERT EINSTEIN: L'umanita' avra' la sorte che sapra' meritarsi.#
<G=@MODINO@DI5gji_j"igjnoj`mnnj\g`\`i\djoo\m`)
ALBERT EINSTEIN: a caso
[ERRORE] letto a invece di I
```

Obiettivo:

L'obiettivo della verifica è di realizzare un programma che, usando le pipe, legga la frase cifrata (dalla pipe di output `-o`) e risponda con la frase in chiaro (sulla pipe di input `-i`). Il programma deve decifrare tutte le frasi proposte (cifrate con chiavi differenti) fino ad ottenere la stampa

```
[*] Congratulazioni! Hai superato il crypto quiz!
```

Suggerimenti:

- Il fatto che le frasi inizino allo stesso modo permette di **calcolare banalmente la chiave** (è sufficiente la prima lettera `'A'`!);
- Non è necessario *bufferizzare* la frase che leggete dalla pipe e costruire una stringa in quanto potete usare direttamente l'altra pipe come *buffer*;
- Ricordatevi che i **char** non sono altro che byte, quindi `'A'+1` è `'B'`;
- Il programma crypto **crea** e **distrugge** le pipe. Quando dovete testare la vostra soluzione, invocate **prima crypto poi il vostro programma**. Esempio:
  `./crypto -i /tmp/pipeIn -o /tmp/pipeOut & (sleep 1; ./soluzione)`
- Se necessario, rendere eseguibile il programma utilizzando `chmod +x crypto`.

NOTA BENE:

- Programmi che **non compilano** o **non superano il test** non verranno valutati (non consegnateli).
- Solo i programmi funzionanti **verranno valutati in base ai commenti.** Commentate in maniera appropriata il vostro programma e **inserite un commento iniziale** in cui spiegate l'idea risolutiva e specificate in che modo dobbiamo invocare il programma crypto per testare la vostra soluzione. Soluzioni non commentate **non** saranno valutate.
- **NON COPIATE E NON FATE COPIARE!** (Nel caso di soluzioni copiate verrà annullata la verifica di chi copia e chi ha fatto copiare indistintamente)

Nascondi soluzione

```c
/*
 * Soluzione di crypto per il corso di Sistemi Operativi 2020
 * Prima verifica: pipe
 *
 * NOTA: Eseguire in questo modo per evitare che parta prima che crypto
 * abbia creato le pipe (in alternativa lanciare i due programmi su due
 * terminali diversi)
 *
 * ./crypto -i /tmp/pipeIn -o /tmp/pipeOut & (sleep 1; ./soluzione )
 *
 * Author: Riccardo Focardi
 *
 * Commento generale (NECESSARIO per una valutazione positiva):
 *
 * La soluzione proposta consiste in un ciclo while(1) che svolge
 * le seguenti operazioni:
 *
 * 1. Legge dalla pipe di output del programma crypto il primo carattere,
 *    che sa essere 'A' (tutte le frasi iniziano con ALBERT EISTEN).
 *    Se la read ritorna 0 esce: questo avviene quando crypto finisce
 *    di mandare frasi e chiude la pipe di output;
 * 2. Calcola la chiave key sottraendo da 'A' il carattere letto c.
 *    Infatti, c = 'A' - key e quindi key = 'A' - c. Poiché i char sono
 *    byte questa operazione aritmetica si puo' fare direttamente tra
 *    variabili di tipo char;
 * 3. Invia 'A' sulla pipe di input di crypto;
 * 4. Legge il resto della frase un carattere alla volta, lo decifra
 *    sommando key e lo invia sulla pipe di input di crypto. Quando il
 *    carattere decifrato e' '#' esce dal ciclo e si prepara a leggere
 *    la frase successiva (torna al punto 1).
 *
 * Si noti che tutte le letture/scritture su pipe avvengono carattere per
 * carattere. Questo ci permette di gestire la decifratura e l'invio a
 * crypto facilmente evitando la creazione di stringhe.
 *
```

```c
 * Si noti inoltre che per uscire correttamente dal ciclo del punto 4
 * si deve testare che il carattere DECIFRATO sia '#' e si deve inoltre
 * inviare '#' sulla pipe di input di crypto. Per questa ragione verra'
 * usato un ciclo do-while che permette, appunto, di verificare la
 * condizione di uscita alla fine del blocco di codice.
 */

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define PIPEIN  "/tmp/pipeIn"
#define PIPEOUT "/tmp/pipeOut"

int main() {
    int fd0,fd1;    // descrittori per le pipe
    char c,d,key;   // variabili per lettura, scrittura e chiave

    /*
     * Apre PIPEOUT in lettura e PIPEIN in scrittura. Se una delle
     * due operazioni di apertura fallisce stampa un messaggio di
     * errore e termina l'esecuzione
     */
    if ( (fd0 = open(PIPEOUT,O_RDONLY))<0 || (fd1 =
open(PIPEIN,O_WRONLY))<0) {
        perror("Errore apertura pipe");
        exit(1);
    }

    while(1) {
        /*
         * Legge un carattere che verrà usato per calcolare la chiave
         * in quanto tutte le frasi cominciano per 'A'.
         * Se la read ritorna 0 significa che crypto ha chiuso la pipe
         * e il programma esce.
         */
        if ( !read(fd0,&c,1) ) {
            exit(0); // pipe chiusa: esce
        }

        /*
         * Calcola la chiave sottraendo ad 'A' il carattere letto.
         * Poiche' la cifratura avviene per sottrazione sappiamo che
         * c = 'A'-key da cui otteniamo key = 'A'-c.
         */
        key = 'A'-c;

        /*
         * Manda il carattere in chiaro 'A' a crypto. Attenzione: la
         * write vuole un puntatore a char quindi è necessario salvare
```

```
88.             * 'A' su una variabile char d e passare &d alla write.
89.             */
90.            d = 'A';             // mette 'A' nella variabile char d
91.            write(fd1,&d,1);     // invia 'A' sulla pipe PIPEIN
92.
93.            /*
94.             * legge da PIPEOUT fino al carattere (decifrato) '#'.
95.             * Ogni carattere letto viene decifrato al volo e scritto
96.             * immediatamente su PIPEIN in modo da non dover costruire
97.             * una stringa. Di fatto PIPEIN funge da buffer.
98.             */
99.            do {
100.               read(fd0,&c,1);  // legge il carattere cifrato c
101.               d = c+key;       // decifra c e salva in d la decifratura
102.               write(fd1,&d,1); // invia d su PIPEIN
103.           } while (d != '#');  // esce quando il carattere è '#'
104.       }
105.   }
```