# Informazione

Autore: **Daniil Radchanka**

Email: **daniil.radchanka@unifi.it**

Matricola: **7079901**

Data di consegna: **10.02.2023**

# Descrizione

In generale si può suddividere tutti le funzioni in due categorie:

1. Funzioni principali (main, ogni funzione di cifratura e decifratura)
2. Funzioni utili (trasformazione di un numero in stringa, controllo del tipo di carattere e etc.)

## *Main (Principale)*

Il programma inizia il suo funzionamento dalla funzione *main*, dove scannerizzando caratteri di *mycypher* applichiamo corrispondente algoritmo di cifratura. Dopo di aver applicato ogni algoritmo di cifratura si procede in ordine inverso applicando algoritmi di decifratura. Per ogni passo di cifrature e decifratura si stamperà sul video la stringa encryptedtext con il new line (utilizzando i metodi per stampare la stringa e per stampare il new line).

## *Funzioni di stampa su video (Utile)*

Aiutano a stampare sul video numeri, stringhe e new line su video restituendo a0 non modificato, che server per il "method chaining", cioè per poter usare questi funzioni in modo seguente:

1. Chiama la funzione che restituisce la stringa in a0
2. Chiama la funzione per stampare la stringa (senza assicurarsi che in a0 c'è la stringa ottenuta dopo 1° passo)
3. Chiama la funzione per stampare il new line (senza assicurarsi che in a0 c'è la stringa ottenuta dopo 1° passo)

## *Funzione di modulo (Utile)*

Se il dividendo e negativo, allora sommo divisore al dividendo finche il dividendo non diventa positivo o uguale a zero. Altrimenti finche il dividendo e maggiore di divisore, devo sottrare il divisore dal dividendo in modo che alla fine il dividendo sarà in intorno tra 0 e divisore-1 (incluso).

## *Funzioni di controllo di appartenenza del carattere ad un insieme (isNumber, isLetter e etc.) (Utile)*

Queste funzioni controllano utilizzando la funzione isBetween il codice del carattere e restituiscono 1 se:

1. IsNumber – se il codice e tra 48 e 57 (incluso)
2. IsCapitalLetter – se il codice e tra 65 e 90 (incluso)
3. IsSmalLetter – se il codice e tra 97 e 122 (incluso)
4. IsLetter – se uno dei due metodi IsCapitalLetter o IsSmallLetter restituisce true (1)
5. IsBetween – restituisce true (1) se a0 sta tra a1 e a2 (incluso).

## *FindNextLetterOccurenceInTheString (Utile)*

Utilizzando l'indirizzo e lunghezza della stringa, si inizia scorrere la stringa dall'inizio di indirizzo sommato con le offset finché non si finisce la stringa oppure non si trova la posizione del carattere cercato. Restituisce in tal caso l'indice di stringa dove e trovato il carattere, se il carattere non era trovato restituisce -1.

## *Length (Utile)*

Scorrendo la stringa finche non si trova il terminatore si conta i caratteri. Restituisce il risultato del questo conto, cioè la lunghezza della stringa.

## *CopyString (Utile)*

Coppia la stringa da a1 in indirizzo a0, sottoscrivendo quello che era in e dopo di a0.

## *StringToInteger (Utile)*

Scorri la stringa e per ogni numero sommi lo a un sommatore, che ogni passo sarà moltiplicato per 10, cioè via via si costruisce il numero di caratteri presenti in stringa.

## *IntegerToString (Utile)*

Finche il numero sia maggiore di zero, si calcola il modulo 10 e si mette tal risultato nella stringa risultante, dopo aver messo il risultato nella stringa si deve dividere il numero per 10. Alla fine otteniamo una stringa che rappresenta il numero invertito (cioè che si ottiene leggendo lo da destra a sinistra). Allora per ottenere il numero originale si mette il terminatore della fine di stringa e si inverte la stringa risultante (utilizzando inverseString), facendo così otteniamo il nostro risultato.

## *Cifratura e decifratura di cesare (Principale)*

Scorriamo la stringa e utilizzando le funzioni isSmallLetter o isCapitalLetter applichiamo seguente formule per la cifratura (dove il numero risultante è il codice in ASCII):

- isSmallLetter – 97+((codice-97+sostK) %26)
- isCapitalLetter - 65+((codice-65+sostK)%26)
- Altrimenti non si applica la cifratura

Dove “sostK” è variabile definita prima per quell’algoritmo e “codice” è il codice di carattere che ora stiamo cifrando.

Per la decifratura dobbiamo invece sottrarre il “sostK” e il resto rimane invariato.

## *Cifratura a blocchi (Principale)*

Fissiamo due indici che utilizzeremo per scannerizzare le nostre stringhe i (per quella che dobbiamo cifrare) e j (quella che utilizzeremo per la cifratura). Indice j dovrà essere annullato appena sarà fuori di lunghezza della stringa per la cifratura. Indice i invece sarà usato proprio per scorrere la stringa per cifrare. Tal indici dovranno essere incrementati insieme in modo che per ogni carattere i-esimo della stringa da cifrare si applicherà seguente formula:

codiceDopoCifratura = 32 + (A + B) % 96;

Dove A è il codice prima di cifratura di i-esimo carattere di stringa da cifrare e B è il codice di j-esimo carattere di stringa utilizzata per la cifratura(blockKey)

## *Decifratura a blocchi (Principale)*

Per la decifratura il procedimento è analogo alla cifratura a blocchi, con la differenza in formula:

codiceDopoDecifratura = (A-32-B) % 96;

Dove A è il codice prima di decifratura di i-esimo carattere di stringa da cifrare e B è il codice di j-esimo carattere di stringa utilizzata per la cifratura(blockKey)

Si può vedere subito che codiceDopoDecifratura in tal formula sarà uguale al modulo 96 di carattere originale, cioè se il carattere prima di cifratura aveva il codice tra 96 e 127, dopo applicazione di cifratura e decifratura si arriverà a un valore in intervallo tra 0 e 32.

Per risolvere questo problema, basta controllare se il valore ottenute è tra 0 e 32 e in tal caso dobbiamo sommare 96 al codice di carattere decifrato, per poter ottenere il codice di carattere originale.

## *Cifratura per occorrenze (Principale)*

La funzione di cifratura e decifratura per occorrenze aveva la difficolta perché era l'unica che non "spostava" il carattere per una formula, ma aveva una nuova interpretazione per ogni carattere incontrato mentre si applicava tal algoritmo.

La mia idea di cifratura per occorrenze era seguente:

- Allochiamo la stringa risultante subito dopo la stringa originale (encryptedtext).
- Per ogni carattere troviamo suoi prossimi occorrenze e marchiamo tal occorrenze in qualche modo per sapere che questi caratteri erano già scelti per la cifratura.
- Per ogni occorrenza di un carattere salviamo la sua posizione in riga risultante con il simbolo "-" precedente.
- Quanto tutti occorrenze erano salvati proseguiamo con il carattere scelto e mettiamo lo spazio in stringa risultante.
- Quando i caratteri della parte originale sono finiti copiamo la stringa risultane in posizione nella stringa originale.

## *Decifratura per occorrenze (Principale)*

La mia idea di decifratura per occorrenze era seguente:

- Allochiamo la riga risultante subito dopo la riga originale (encryptedtext).
- Ricordiamo il carattere in stringa originale per poter metterla nella stringa risultante.
- Scannerizziamo la stringa finché non si trova lo spazio e ogni volta trovato "-" prendiamo il numero seguente dopo "-", convertiamo tal numero dalla stringa a un numero e scriviamo il carattere salvato in posizione di tal numero.
- Dopo di aver preso tutti i numeri del carattere si prosegue con il carattere che viene subito dopo lo spazio.
- Alla fine copiamo la stringa risultante in posizione della stringa originale.

## *Cifratura e decifratura con dizionario (Principale)*

Per la cifratura scorriamo la stringa e per ogni carattere con il suo codice applichiamo questa cifratura (dove il numero risultante è il codice in ASCII):

- Se è una lettera minuscola –90-(codice-97)
- Se è una lettera maiuscola – 122-(codice-65)
- Se è un numero – 57-(codice-48)

Per la decifratura basta applicare un'altra volta la cifratura a dizionario.

## *InverseString*

Questa funziona si può descrivere in seguente pseudocodice:

```
String inverseString (String str) {
    int length = length(str);
    int i = 0;
    int j = length-1;
    length = length >> 1;
    while (length > 0) {
```

```
            swap characters in str with pos i and j;
            i=i+1;
            j=j-1;
            length=length-1;
        }
        return str;
}
```

## *L'uso dei registri e memoria*

Dal punto di vista della memoria e l'uso dei registri tramite l'uso classico per invocazione di un metodo si deve osservare solo l'importanza della posizione di encryptedtext, che deve essere obbligatoriamente dichiarato per ultimo. Questa variabile cambierà la sua lunghezza dopo l'applicazione di cifratura e decifratura per occorrenze e se non sarà dichiarata per ultimo tal funzioni di cifratura scriveranno i suoi risultati nei prossimi variabili dopo di *encryptedtext.*

*Avevo deciso di fare cosi, perché pensavo di due implementazioni di tale restituzione della stringa cifrata:*

- Variabile globale (quello che ho implementato)
- Variabile di ritorno dalle funzioni;

Variabile di ritorno sarebbe più complicata per implementazione perché dovrò gestire il ritorno di tale stringa e il suo passaggio dento i metodi. Quindi variabile globale è una soluzione semplice e veloce per questo problema.

## Test di corretto funzionamento

### Test 1

***Data***

```
mycypher: .string "ABCDE"
myplaintext: .string "Tests"
sostK: .word -108
blockKey: .string "aA-Bb/"
```

***Result***

```
Tests
Paopo
qb\r1
q-1 b-2 \-3 r-4 1-5
J-8 Y-7 \-6 I-5 8-4
4-8 5-I 6-\ 7-Y 8-J
J-8 Y-7 \-6 I-5 8-4
q-1 b-2 \-3 r-4 1-5
qb\r1
Paopo
Tests
```

# Test 2

***Data***

```
mycypher: .string "AABBCCDDE"
myplaintext: .string "te/sT"
sostK: .word -108
blockKey: .string "aB-"
```

***Result***

```
te/sT
pa/oP
lw/kL
-y|,N
N{iMP
N-1 {-2 i-3 M-4 P-5
N-1 --2-6-10-14-18 1-3  -4-8-12-16 {-5 2-7 i-9 3-11 M-13 4-15 P-17 5-19
m-8 --7-3-89-85-81 8-6  -5-1-87-83 {-4 7-2 R-0 6-88 n-86 5-84 k-82 4-80
N-1 --2-6-10-14-18 1-3  -4-8-12-16 {-5 2-7 i-9 3-11 M-13 4-15 P-17 5-19
91-5 71-P 51-4 31-M 11-3 9-i 7-2 5-{ 61-21-8-4-  3-1 81-41-01-6-2-- 1-N
N-1 --2-6-10-14-18 1-3  -4-8-12-16 {-5 2-7 i-9 3-11 M-13 4-15 P-17 5-19
m-8 --7-3-89-85-81 8-6  -5-1-87-83 {-4 7-2 R-0 6-88 n-86 5-84 k-82 4-80
N-1 --2-6-10-14-18 1-3  -4-8-12-16 {-5 2-7 i-9 3-11 M-13 4-15 P-17 5-19
N-1 {-2 i-3 M-4 P-5
N{iMP
-y|,N
lw/kL
pa/oP
te/sT
```

# Test 3

## *Data*

```
10 mycypher: .string "EDCBA"
11 myplaintext: .string "simple test -1"
12 sostK: .word 5
13 blockKey: .string "104A-"
```

## *Result*

```
simple test -1
1- tset elpmis
8- GHVG VOKNRH
8-1 --2  -3-8 G-4-7 H-5-14 V-6-9 O-10 K-11 N-12 R-13
)}%!z~"t!z$},!4~$!8m9}).~%pJ.#~)tPz" tLz"!tOz""tSz"#
)}%!e~"y!e$},!4~$!8r9}).~%uO.#~)yUe" yQe"!yTe""yXe"#
)}%!z~"t!z$},!4~$!8m9}).~%pJ.#~)tPz" tLz"!tOz""tSz"#
8-1 --2  -3-8 G-4-7 H-5-14 V-6-9 O-10 K-11 N-12 R-13
8- GHVG VOKNRH
1- tset elpmis
simple test -1
```