Progetto ARCHITETTURE DEGLI ELABORATORI 2018-2019

Simone Cappabianca - Mat: 5423306
simone.cappabianca@stud.unifi.it

Agosto 18, 2019

Contents

1 Descrizione

Lo scopo del progetto e' quello di cifrare e decifrare un dato messaggio in base ad una data chiave di cifratura.

1.1 Programma

Per la realizzazione del progetto sono state fatte le seguenti scelte:

- i percorsi dei file *chiave.txt*, *messaggio.txt*, *messaggioCifrato.txt* e *messaggioDecifrato.txt* sono tutti path assoluti, questo perché su Mac OS ho riscontrato problemi con i path relativi;
- i buffer utilizzati all'interno del programma per contenere i messaggi sono definiti come *.space 1024* in questo modo non dovrebbero insorgere problemi di spazio con l'algoritmo **E**, non riuscento a stimare con precisione l'occupazione massima di memoria che puó richiedere la cifratura con l'agoritmo **E** ho preferito sovrastimarla;
- prima di eseguire il programma é necessario assicurarsi che i file *messaggioCifrato.txt* e *messaggioDecifrato.txt* siamo presenti nei path indicati nel codice del programma e che siamo vuoti, in questo modo si carantisce la corretta eseguzione del processo di crifratura e di decifratura.

1.1.1 main

La procedura **main** esegue in ordine le seguenti operazioni:

1. leggere la chiave di cifratura controllando che il file *chiave.txt* esista e nell'eventualitá in cui il controllo fallisse interrompere l'esecuzione del programma, per quanto riguarda la corretezza semantica della chiave non viene fatto alcun tipo di controllo quindi si assume che la chiave appartenga all'insime delle disposizioni con ripetizione dei 5 elementi (A,B,C,D,E) raggrupati 4 a 4;
2. leggere il messaggio originale su cui applicare gli algoritmi di cifratura

e di controllare che il file *messaggio.txt* esista e che il file non sia vuoto nel eventualitá in cui uno o entrambi i controlli fallissero interrompere l'esecuzione del programma, per quanto riguarda il contenuto del messaggio non viene fatto alcun tipo di controllo;

3. cifrare il messaggio originale contenuto all'interno del file *messaggio.txt* utilizzando la chiave di cifratura contenuta in *chiave.txt*, tale chiave oltre a stabilire quali algoritmi di cifratura utilizzare definisce anche la loro seguenza di applicazione, infine salva il risultato della cifratura all'interno del file *messaggioCifrato.txt*;

4. decifrare il messaggio cifrato contenuto all'interno del file *messaggioCifrato.txt* utlizzando sempre la chiave di cifratura contenuta all'interno del file *chiave.txt* per determinare queli algoritmi di decifratura utilizzare e la loro seguenza di applicazione che é l'inversa rispetto a quella di cifratura, infine salvare il messaggio decifrato all'interno del file *messaggioDecifrato.txt*.

1.1.2 encryptMsg

La procedura *encryptMsg* prende come parametri in ingresso la chiave di cifratura(*$a0*) e la sua lunghezza(*$a1*) e restituisce la lunghezza del messaggio cifrato(*$v0*). Per quanto riguarda gli algoritmi di cifratura **A**,**B**,**C**,**D** la procedura ritornerá come lunghezza la stessa lunghezza del messaggio originale mentre nel caso dell'algoritmo **E** ritornerá sicuramente un valore superiore alla lunghezza del messaggio originale.

Per applicare nella giusta sequenza gli algoritmi di cifratora é stato realizzato un ciclo che scandisce la chiave di cifratura dal primo all'ultimo carattere e con l'ausilio di una *JumpAddressTable* é stato realizzato uno switch che in base al carattere selezionato della chiave richiama il giusto algoritmo di cifratura. La *JumpAddressTable* é stata definita nel seguente modo:

```
jumpEncrtpTable: .word  encrptA encrptB encrptC encrptD encrptE
```

La *JumpAddressTable* definita sopra non é altro che un *Array* di *Word* e questo significa che nella posizione di indice 0 c'é la parola *encrptA* e cosi via fino all'indice 4 che contine la parola *encrptE*.

Per poter utilizare la *jumpEncrtpTable* ad ogni carattere letto dalla chiave di cifratura é necessario sottrarre 65 dal corrispettivo valore ascii come si puó vedere nel codice sottostante.

```
lb $t4,($t3)
beqz $t5,endEncrpt
sub $t0,$t4,65
mul $t0,$t0,4
add $t0,$t0,$t1
lw $t0,0($t0)
jr $t0
```

In questo modo con il carattere **A**, che corrisponde al valore *Ascii* 65(Dec), viene recuperata l'etichetta *encrptA* che fa saltare direttamente codice sottostante che applica l'algoritmo di cifratura corrispondente alla lettera **A**.

```
encrptA:
  addi $sp,$sp,-4
  sw $t0,0($sp)
  addi $sp,$sp,-4
  sw $t1,0($sp)
  addi $sp,$sp,-4
  sw $t3,0($sp)
  addi $sp,$sp,-4
  sw $t4,0($sp)
  addi $sp,$sp,-4
  sw $t5,0($sp)
  la $a0,bufferMsgData
  jal encryptA
  lw $t5,0($sp)
  addi $sp,$sp,4
  lw $t4,0($sp)
  addi $sp,$sp,4
  lw $t3,0($sp)
  addi $sp,$sp,4
  lw $t1,0($sp)
  addi $sp,$sp,4
  lw $t0,0($sp)
  addi $sp,$sp,4
  j exitCaseEncrpt
```

Cosí facendo in base alla composizione della chiave di cifatura si stabilisce quale algoritmi applicare e il loro ordine di applicazione in modo tale che al termine dell'esecuzione della procedura il buffer *bufferMsgData* conterrá il messeggio cifrato.

1.1.3 decryptMsg

La procedura *decryptMsg* prende come parametri in ingresso la chiave di cifratura(*$a0*) e la sua lunghezza(*$a1*) e restituisce la lunghezza del messaggio cifrato(*$v0*). In realtá per tutti gli algoritmi di decifratura **A**,**B**,**C**,**D** e **E** la procedura ritornerá sempre la stessa lunghezza che coinciderá con lunghezza del messaggio originale.

Per applicare nella giusta sequenza gli algoritmi di decifratora é stato realizzato un ciclo che scandisce la chiave di cifratura dall'ultimo al primo carattere in modo tale che gli algoritmi di decifratura siamo richiamti nell'ordine inverso rispetto a quelli di cifratura. Con l'ausilio di una *JumpAddressTable* é stato realizzato uno switch che in base al carattere selezionato della chiave richiama il giusto algoritmo di decifratura. La *JumpAddressTable* é stata definita nel seguente modo:

```
jumpDecrptTable: .word  decrptA decrptB decrptC decrptD decrptE
```

La *JumpAddressTable* definita sopra non é altro che un *Array* di *Word* e questo significa che nella posizione di indice 0 c'é la parola *decrptA* e cosi via fino all'indice 4 che contine la parola *decrptE*.

Per poter utilizare la *jumpDecrtpTable* ad ogni carattere letto dalla chiave di cifratura é necessario sottrarre 65 da corrispettivo valore ascii come si puó vedere nel codice sottostante.

```
lb $t4,($t3)
bltz $t5,endDecrpt
sub $t0,$t4,65
mul $t0,$t0,4
add $t0,$t0,$t1
lw $t0,0($t0)
jr $t0
```

In questo modo con il carattere **A**, che corrisponde al valore *Ascii* 65(Dec), viene recuperata l'etichetta *decrptA* che fa saltare direttamente codice sottostante che applica l'algoritmo di decifratura corrispondente alla lettera **A**.

```
decrptA:
  addi $sp,$sp,-4
  sw $t0,0($sp)
  addi $sp,$sp,-4
  sw $t1,0($sp)
  addi $sp,$sp,-4
  sw $t3,0($sp)
  addi $sp,$sp,-4
  sw $t4,0($sp)
  addi $sp,$sp,-4
  sw $t5,0($sp)
  la $a0,bufferEncrptData
  jal decryptA
  lw $t5,0($sp)
  addi $sp,$sp,4
  lw $t4,0($sp)
  addi $sp,$sp,4
  lw $t3,0($sp)
  addi $sp,$sp,4
  lw $t1,0($sp)
  addi $sp,$sp,4
  lw $t0,0($sp)
  addi $sp,$sp,4
  j exitCaseDecrpt
```

Cosí facendo in base alla composizione della chiave di cifatura si

stabilisce quale algoritmi applicare e il loro ordine di applicazione in modo tale che al termine dell'esecuzione della procedura il buffer *bufferEncrptData* conterrá il messeggio decifrato.

1.1.4 encryptA

La procedura *encryptA* realizza l'algoritmo di cifratura **A**. Ha come unico parametro il messaggio da cifrare(*$a0*) e restituisce la lunghezza del messaggio cifrato(*$v0*). Questo algoritmo non é altro che un **Cifrario di Cesare** e non altare la lunghezza del messaggio ma solo i caratteri contenuti al suo interno, ad esempio sostituira la lettera H con la lettera L ma non modificherá in alcun modo il numero di caratteri di cui é composto il messaggio.

Per realizzare questo algoritmo é stato utilizzato un ciclo per scandire ogni singolo carettere di cui é composto il messaggio da cifrare. Ogni singolo carattere del messaggio viene sostituito con il **modulo** in base 256 del suo corrispettivo valore $Ascii$ incrememtato di 4 come si puó vedere dal codice sottostante.

```
move $a0,$t1                    # Start - char encoding
addi $a0,4                      # .
li $a1,256                      # .
jal module                      # End - char endcoding
```

L'utilizzo della funzione modulo é necessaria per garantire che il nuovo carattere sia uno dei 256 caratteri definiti nella codifica **Ascii Estesa**. Per la realizzazione di questo algoritmo di cifratura é stato necessario definire una prodocura *module*.

```
########################################
#                module                #
########################################
module:
  # Procedure to calcolate a mod b
  # $a0: number
  # $a1: module
  # $v0: result of number mod modulo

  div $a0,$a1
  mfhi $v0
  jr $ra
```

Questa procedura di servizio prende come parametri in numero di cui fare il modulo(*$a0*) la base del modulo(*$a1*) e restituisce il modulo(*$v0*). Per calcolare il modulo prima si effetua la divisione del numero per la base utilizzando l'istruzione *div* e poi utiliziamo l'istruzione *mfhi* per recuperare il resto della divisione che coindice con il modulo.

1.1.5 decryptA

La procedura *decryptA* realizza l'algoritmo di decifratura di un messaggio cifrato con l'algoritmo di cifratura **A**. Ha un unico parametro il messaggio da decifrare(*$a0*) e restituisce la lunghezza del messaggio decifrato(*$v0*).Questa procedura é identica alla produra *encryptA* ad esclusione del fatto che invece di sommare 4 al corrispondente valore *Ascii* del carattere selezionato viene sottratto 4 come si puó vedere nel codice sottostante.

```
move $a0,$t1                    # Start - char decoding
addi $a0,-4                     # .
li $a1,256                      # .
jal module                      # End - char decoding
sb $v0,0($t0)                   # Store decoded char
```

1.1.6 encryptB

Questa procedura é uguale alla procedura *encryptA* ha come unico parametro d'ingresso il messaggio da cifrare(*$a0*) e restituisce la lunghezza del messaggio cifrato(*$v0*) e implementa come l'agoritmi di cifratura sempre un **Cifrario di Cesare** l'unica differenza e viene applicato solamente a caratteri la cui posizione all'interno del messaggio é pari.

Per stabilire se la posizione del carattere e pari abbiamo utilizzato la procedura *module* definita e giá utilizzata per procedura *encryptA*. Nel caso specifico come parametri alla procedura gli viene passata la posizione del carattere e 2, se la procedura resituisce 0 allora la posizione e pari e quindi il carattere viene cifrato altrimenti viene lasciato inalterato. Il codice sotto stante mostra l'utilizzo delle funzione *modulo* per daterminare se la posizione é pari.

```
move $a0,$t2
li $a1,2
jal module                      # Calcolated the module the
                                # position in base 2
move $t3,$v0
lw $v0,0($sp)
add $sp,$sp,4
lw $t2,0($sp)
add $sp,$sp,4
lw $t1,0($sp)
add $sp,$sp,4
lw $t0,0($sp)
add $sp,$sp,4
bnez $t3,jumpEncodingB          # Chech if the position is even
add $sp,$sp,-4
sw $t0,0($sp)
add $sp,$sp,-4
sw $t2,0($sp)
add $sp,$sp,-4
sw $v0,0($sp)
```

```
  move $a0,$t1                    # Start - char encoding
  addi $a0,4                      # .
  li $a1,256                      # .
  jal module                      # End - char encoding
  move $t1,$v0
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t2,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
jumpEncodingB:
  sb $t1,0($t0)                   # Store coded char
```

1.1.7 decryptB

La procedura *decryptB* realizza l'algoritmo di decifratura di un messaggio cifrato con l'algoritmo di cifratura **B**. Ha un unico parametro il messaggio da decifrare(*$a0*) e restituisce la lunghezza del messaggio decifrato(*$v0*).Questa procedura é identica alla produra *encryptB* ad esclusione del fatto che invece di sommare 4 al corrispondente valore *Ascii* del carattere selezionato viene sottratto 4 come si puó vedere nel codice sottostante.

```
  move $a0,$t1                    # Start - char decoding
  addi $a0,-4                     # .
  li $a1,256                      # .
  jal module                      # End - char decoding
```

1.1.8 encryptC

Questa procedura é identica alla procedura *encryptB* solamente che in questa i caratteri che vengono cifrati sono quelli la cui posizione all'interno del messaggio é dispari.

1.1.9 decryptC

La procedura *decryptC* realizza l'algoritmo di decifratura di un messaggio cifrato con l'algoritmo di cifratura **C**. Ha un unico parametro il messaggio da decifrare(*$a0*) e restituisce la lunghezza del messaggio decifrato(*$v0*).Questa procedura é identica alla produra *encryptC* ad esclusione del fatto che invece di sommare 4 al corrispondente valore *Ascii* del carattere selezionato viene sottratto 4 come si puó vedere nel codice sottostante.

```
  move $a0,$t1                    # Start - char decoding
  addi $a0,-4                     # .
```

```
li $a1,256                          # .
jal module                          # End - char decoding
```

1.1.10 encryptD

La procedura *encryptD* realizza l'algoritmo di cifratura **D**. Ha come unico parametro il messaggio da cifrare(*$a0*) e restituisce la lunghezza del messaggio cifrato(*$v0*). Questo algoritno deve invertire l'ordine dei caratteri del messaggio. Ad esempio se il messagio contiene caratteri *Hello!!!* l'argoritmo deve restituire *!!!olleH*.

Per realizzare questo algoritmo e stato utilizato un ciclo per scorrere il messaggio contemporaneamente dalla sua testa verso la su coda(*$t0*) e dalla sua coda alla sua testa(*$t1*) lo scambio dei caratteri viene effettuato fino a quando non si raggiunge la meta del messaggio come si puó vedere dal codice sottostante.

```
nextChrEncryptD:
  lb $t3,($t0)
  lb $t4,($t1)
  sub $a3,$t1,$t0
  blez $a3,endBufferEncryptD        # Check the half of the message
  sb $t4,($t0)                      # Store char in new position
  sb $t3,($t1)                      # Store char in new position
  addi $t0,$t0,1
  addi $t1,$t1,-1
  j nextChrEncryptD
```

Per capire quando il ciclo si deve interrompere viene fatta la sottazione dei *$t1* meno *$t0* che contengono rispettivamente la popsizione dell'ultimo carattere letto a partire dal fondo del messaggio e la posizione dell'ultimo carattere letto a partire dall'inzio del messaggio. Se il risultato della loro sottrazione é maggiore di zero significa che i due indici non si sono incontrati e quindi si possono scambiare i caratteri. Se la sottrazione da come risultato zero questo significa la lunghezza del messaggio é dispari e che antrambi *$t0* e *$t1* sono sul carattere centrale, se la sottrazione é minore di zero vuol dire che la lunghezza del messaggio é pari e che i due indici si sono incrociati, in entrambi i casi il ciclo deve essere interrotto e di conseguenza anche la cifratura del messaggio.

1.1.11 decryptD

La procedura *decryptD* realizza l'algoritmo di decifratura di un messaggio cifrato con l'algoritmo di cifratura **D**. Ha come unico parametro il messaggio da decifrare(*$a0*) e restituisce la lunghezza del messaggio decifrato(*$v0*). Da come si puó vedere dal codice sottostante questa procedura non fa altro che chiamare la procedura *encryptD* passandogli come parametro il messaggio precedentemente cifrato con l'algoritmo D.

```
addi $sp,$sp,-4
sw $ra,0($sp)
jal encryptD
lw $ra,0($sp)
addi $sp,$sp,4
jr $ra
```

Questo é possibile per la particolare natura di questo algoritmo, se viene applicato su un testo un numero di volte dispari allora il testo avrá l'ordine dei caratteri invertivo, se invece viene applicato un numero di volte pari il testo rimanne invariato.

1.1.12 encryptE

La procedura *encryptE* realizza l'algoritmo di cifratura **E**. Ha come unico parametro il messaggio da cifrare(*$a0*) e restituisce la lunghezza del messaggio cifrato(*$v0*). Questo algoritmo deve cifrare il messaggio mostrando per ogni singolo carattere tutte le posizioni delle sue occorrenze separate da **-**, tra un carattere e l'altro deve essere inserito uno spazio. Ad esempio se il messaggio contine i seguenti caratteri *Hello!!!* l'algoritmo deve restituire *H-0 e-1 l-2-3 0-4 !-5-6-7*.

Per realizzare questo algoritmo come prima cosa é necessario determinare di quali caratteri é composto il messaggio. Per tenera traccia di tutti i caratteri di cui é composto il messaggio viene utilizzata una lista puntata in cui vengono inseriti i caratteri la prima volta che vengono incontrati. É stata scelta una lista puntata per non é possibile sapere a priori il numero dei caratteri presenti all' interno del messaggio quindi é stata scelta una struttura dati dinamica. Per costruire la lista é stata definita la procedura di servizio *sbrk* che come potete vedera da codice sottostante non é altro che un wrapper alla chiamata *syscall* con il registro *$v0* uguale a 9 per allocare *heap memory*.

```
#########################################
#                 sbrk                  #
#########################################
sbrk:
  # Procedure to allocate heap memory
  # $a0: number of bytes to allocate
  # $v0: address of allocated memory

  li $v0,9
  syscall
  jr $ra
```

Questa procedura ha come paramentro il numero di bytes da allocare(*$a0*) e restituisce l'indirizzo della memoria allocata(*$v0*). Nel progetto viene sempre chiamata passandoglio come parametro 5 in questo perché primi 4 bytes servono per linkare la cella successiva ed il quinto byte serve per contenere il carattere.

Per la costruzione della lista dei caratteri viene utilizzato un ciclo che scandisce ogni carattere del messaggio e per ognuno di questi controlla se é giá presente all' interno della lista. Nel caso in cui il carattere risulta giá presente nella lista passa al successivo carattere del messaggio, nel caso contrario, cioé é la prima occorrenze del carattere, aggiunge un nuovo elemento(procedura *sbrk*) in coda alla lista per contenere il nuovo carattere.

Una volta costruita la lista con tutti i caratteri presenti all'interno del messaggio viene eseguito un ciclo su di essa e per ogni singolo carattere della lista viene eseguito un ciclo sul messaggio per cerca tutte le posizioni del caratte all'interno del messagio in questo modo si costruisce la stringa di cifratura del carattere. Quando il ciclo sul messaggio termina viene prese il carettere successivo della lista e viene effettuata nuovamente la ricerca delle sue posizioni all'interno del messaggio. L'algoritmo termina quando terminano gli elementi della lista.

1.1.13 decryptE

La procedura *decryptE* realizza l'algoritmo di cifratura di un messaggio cifrato con l'algoritmo di cifratura **E**. Ha come unico parametro il messaggio da decifrare(*$a0*) e restituisce la lunghezza del messaggio decifrato(*$v0*). Per effettuare la decifratura del messaggio cifrato con l'algoritmo **E** é stato necessario utilizzare un *buffer* di servizio in cui copiare il messaggio da decifrare e al tempo stesso cancellare il *buffer* che lo conteneva come si puó

vedre ne codice sottostante.

```
  move $t0,$a0                          # Message
  la $t1,bufferDecrptDataTmp            # Temp buffer
loopCopyDecryptE:                       # Copy the encrypt message in the
  lb $t2,0($t0)                         # temp buffer and clear the original
  beqz $t2,endCopyDecryptE
  sb $t2,0($t1)
  sb $zero,0($t0)
  add $t0,$t0,1
  add $t1,$t1,1
  j loopCopyDecryptE
```

Una volta effettuata la copia del messaggio cifrato per realizzare la decifratura é stato sufficente un ciclo sulla sulla copia e una volta riconosciuti i caratteri e le relative posizioni andarli a posizionare nel all'interno del *buffer* originale.

1.1.14 Procedure di servizio

Per la realizzazione del progetto sono state create tutta una serie di prodecure di servizio come ad esempio ma procedura *module* che é stata utilizzata per gli algoritmi **A**, **B** e **C**. Poi sono state create tutta una serie di prodecure che fanno il *wrapping* di alcune chiamate *syscall* come ad esempio *printStr*, *printInt*, *printChr* e molte altre tra cui anche la *sbrk* utilizzata per la realizzazione dell'algoritmo **E**.

La realizzazione di tutte queste procedure di servizio ha contribuito a migliore suprattutto due aspetti molto importanti del codice, che sono:

- **Modularitá**;
- **Leggibilitá** .

Un codice che soddisfa questi due requisiti é senza dubbio un codice piú semplice da mantenere.

2 Test

Il test ti corretta funzionamento del programma ha esito positivo quando la sua l'escuzione produce un messaggio decifrato contenuto in *messaggioDecifrato.txt* che coincide con il messaggio originale contenuto in *messaggio.txt*.

2.1 Esempio

L'esempio sotto riportata é decisamente semplice per ovviare al problema che il messaggio cifrato (*messaggioCifrato.txt*) prodotto utilizzando gli algoritmi **A**, **B** e **C** proprio per la caratterista della cifratura puó contenere dei caratteri/simboli che non possono essere rappresentati.

`../samples/chiave.txt`

```
AABE
```

`../samples/messaggio.txt`

```
Hello!!!
```

`../samples/messaggioCifrato.txt`

```
T-0 m-1 x-2 t-3 {-4 )-5-7 --6
```

`../samples/messaggioDecifrato.txt`

```
Hello!!!
```

3 Codice

```
################################################################################
# Title: Encrypted messages                    Filename: encrypter.s           #
# Author: Simone Cappabianca                   Date: 21/08/2019                #
# Description: MIPS Assembly project for the course of Architetture degli      #
#              Elaboratori - A.A. 2018/2019 -                                  #
# Input: key of encryption (chiave.txt)                                        #
#        Message to be encryped (massaggio.txt)                                #
# Output: Message encrypetd (messaggioCifrato.txt)                             #
#         Message decrypted (messaggioDecifrato.txt)                           #
################################################################################

################################################################################
################################## DATA SEGMENT ################################
################################################################################
.data

# Key
fullNameOfKey: .asciiz "Users/simonecappabianca/Documents/University/ProjectAE_2018-2019/samples/chiave.txt"
bufferKeyData: .space 5

# Message
fullNameOfMsg: .asciiz "Users/simonecappabianca/Documents/University/ProjectAE_2018-2019/samples/messaggio.txt"
bufferMsgData: .space 1024

# Encrypt Message
fullNameOfEncrptMsg: .asciiz "Users/simonecappabianca/Documents/University/ProjectAE_2018-2019/samples/messagg
bufferEncrptData: .space 1024
bufferEncrptDataTmp: .space 1024

# Decrypt Message
fullNameOfDecrptMsg: .asciiz "Users/simonecappabianca/Documents/University/ProjectAE_2018-2019/samples/messagg
bufferDecrptData: .space 1024
bufferDecrptDataTmp: .space 1024

positionCounter: .byte 0

# Messages
promptReadKey: .asciiz "1 - Read file chiave.txt\n"
promptReadMsg: .asciiz "\n2 - Read file messaggio.txt\n"
promptLengthMsg: .asciiz "\n2.1 - Length of messsage: "
promptEncrptMsg: .asciiz "\n3 - Encrypt the original message\n"
promptWriteEncrptMsg: .asciiz "\n4 - Write the file mesasggioCifrato.txt\n"
promptDecrptMsg: .asciiz "\n5 - Decrypt the encrypted message\n"
promptWriteDecrptMsg: .asciiz "\n6 - Write the file messaggioDecifrato.txt\n"
msgErrKeyNotExist: .asciiz "File of key not exist!\n"
msgErrKeyIsEmpty: .asciiz "File of key is empty!\n"
msgErrMsgNotExist: .asciiz "File of message not exist!\n"
msgErrMsgIsEmpty: .asciiz "File of message is empty!\n"
msgEndProgram: .asciiz "\nEnd program\n"
lineFeed: .asciiz "\n"
promptAlgEncrptA: .asciiz "\nApplied the algorithm of encryption A\n"
promptAlgEncrptB: .asciiz "\nApplied the algorithm of encryption B\n"
promptAlgEncrptC: .asciiz "\nApplied the algorithm of encryption C\n"
promptAlgEncrptD: .asciiz "\nApplied the algorithm of encryption D\n"
promptAlgEncrptE: .asciiz "\nApplied the algorithm of encryption E\n"
promptAlgDecrptA: .asciiz "\nApplied the algorithm of decryption A\n"
promptAlgDecrptB: .asciiz "\nApplied the algorithm of decryption B\n"
promptAlgDecrptC: .asciiz "\nApplied the algorithm of decryption C\n"
promptAlgDecrptD: .asciiz "\nApplied the algorithm of decryption D\n"
promptAlgDecrptE: .asciiz "\nApplied the algorithm of decryption E\n"
positionSeparetor: .asciiz "-"
charSeparetor: .asciiz " "

.align 2
jumpEncrtpTable: .word  encrptA encrptB encrptC encrptD encrptE
jumpDecrptTable: .word  decrptA decrptB decrptC decrptD decrptE

################################################################################
################################## Code segment ################################
################################################################################
.text
.globl main
.align 2
```

```
#########################################
#                printStr               #
#########################################
printStr:
  # Procedure to print a string
  # $a0: address of null-terminated string to print

  li $v0,4
  syscall
  jr $ra

#########################################
#                printInt               #
#########################################
printInt:
  # Procedure to print a integer
  # $a0: integer to print

  li $v0,1
  syscall
  jr $ra

#########################################
#                printChr               #
#########################################
printChr:
  # Procedure to print a char
  # $a0: character to print

  li $v0,11
  syscall
  jr $ra

#########################################
#                printBuffer            #
#########################################
printBuffer:
  # Procedure to print a buffer
  # $a0: address of input buffer

  move $t0,$a0                                  # $t3 buffer
  addi $sp,$sp,-4
  sw $ra,0($sp)
nextChr:
  lb $t1,($t0)
  beqz $t1,endBuffer
  addi $sp,$sp,-4
  sw $t0,0($sp)
  addi $sp,$sp,-4
  sw $t1,0($sp)
  move $a0,$t1
  jal printChr
  lw $t1,0($sp)
  addi $sp,$sp,4
  lw $t0,0($sp)
  addi $sp,$sp,4
  add $t0,$t0,1
  j nextChr

endBuffer:
  lw $ra,0($sp)
  addi $sp,$sp,4
  jr $ra

#########################################
#                readStr                #
#########################################
readStr:
  # Procedure to read a string
  # $a0: address of input buffer
        # $a1: maximum number of characters to read

  li $v0,8
        syscall
  jr $ra
```

```
#########################################
#                length                 #
#########################################
length:
  # Procedure to calcolate length of string
  # $a0: string
  # $v0: length of string

  move $t2,$a0
  move $t1,$zero
nextCh:
  lb $t0,($t2)
  beqz $t0,strEnd
  add $t1,$t1,1
  add $t2,$t2,1
  j nextCh

strEnd:
  add $t1,$t1,-1
  move $v0,$t1
  jr $ra

#########################################
#           openFileToRead              #
#########################################
openFileToRead:
  # Procedure to open file in read mode
  # $a0 = address of null-terminated string containing filename

  li $v0,13                             # system call for open file
  li $a1,0                              # flag for reading
  li $a2,0                              # mode is ignored
  syscall                               # open a file
  jr $ra

#########################################
#                readFile               #
#########################################
readFile:
  # Procedure to read a file
  # $a0: file descriptor
  # $a1: address of input buffer
  # $a2: maximum number of characters to read

  li $v0,14                             # system call for read file
  syscall
  jr $ra

#########################################
#           openFileToWrite             #
#########################################
openFileToWrite:
  # Procedure to open file in write mode
  # $a0: output file name

  li $v0, 13                            # system call for open file
  li $a1, 1                             # flag for writing
  li $a2, 0                             # mode is ignored
  syscall                               # open a file
        jr $ra

#########################################
#               writeFile               #
#########################################
writeFile:
  # Procedure to write a file
  # $a0: file descriptor
  # $a1: address of buffer from which to write
  # $s2: hardcoded buffer length

  li $v0, 15                            # system call for write file
  syscall
  jr $ra

#########################################
#               closeFile               #
```

```
########################################
closeFile:
  # Procedure to close file
  # $a0: file descriptor

  li $v0,16                                # system call for close file
  syscall
  jr $ra

########################################
#                sbrk                  #
########################################
sbrk:
  # Procedure to allocate heap memory
  # $a0: number of bytes to allocate
  # $v0: address of allocated memory

  li $v0,9
  syscall
  jr $ra

########################################
#                module                #
########################################
module:
  # Procedure to calcolate a mod b
  # $a0: number
  # $a1: module
  # $v0: result of number mod modulo

  div $a0,$a1
  mfhi $v0
  jr $ra

########################################
#                encryptA              #
########################################
encryptA:
  # Procedure for algorithm A
  # $a0: original message
  # $v0: length of the encrypted message

  move $t0,$a0                             # $t0: Message
  move $v0,$zero
  add $sp,$sp,-4
  sw $ra,0($sp)
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $v0,0($sp)
  la $a0,promptAlgEncrptA
  jal printStr
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
nextChrEncryptA:
  lb $t1,($t0)
  beqz $t1,endBufferEncryptA
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $t1,0($sp)
  add $sp,$sp,-4
  sw $v0,0($sp)
  move $a0,$t1                             # Start - char encoding
  addi $a0,4                               # .
  li $a1,256                               # .
  jal module                               # End - char endcoding
  move $t2,$v0
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t1,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
```

```
  sb $t2,0($t0)                          # Store coded char
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $v0,($sp)
  move $a0,$t2
  jal printChr
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
  add $t0,$t0,1
  addi $v0,1
  j nextChrEncryptA

endBufferEncryptA:
  lw $ra,0($sp)
  addi $sp,$sp,4
  jr $ra

########################################
#                decryptA              #
########################################
decryptA:
  # Procedure for decryption algorithm A
  # $a0: coded message
  # $v0: length of the decrypted message

  move $t0,$a0                           # $t0: Message
  move $v0,$zero
  add $sp,$sp,-4
  sw $ra,0($sp)
  add $sp,$sp,-4
  sw $t0,0($sp)
  addi $sp,$sp,-4
  sw $v0,0($sp)
  la $a0,promptAlgDecrptA
  jal printStr
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
nextChrDecryptA:
  lb $t1,($t0)
  beqz $t1,endBufferDecryptA
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $t1,0($sp)
  add $sp,$sp,-4
  sw $v0,0($sp)
  move $a0,$t1                           # Start - char decoding
  addi $a0,-4                            # .
  li $a1,256                             # .
  jal module                             # End - char decoding
  sb $v0,0($t0)                          # Store decoded char
  move $a0,$v0
  jal printChr
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t1,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
  add $t0,$t0,1
  addi $v0,1
  j nextChrDecryptA

endBufferDecryptA:
  lw $ra,0($sp)
  addi $sp,$sp,4
  jr $ra

########################################
#                encryptB              #
########################################
```

```
encryptB:
  # Procedure for algorithm B
  # $a0: original message
  # $v0: length of the encrypted message

  move $t0,$a0                          # $t0: Message
  move $v0,$zero
  add $sp,$sp,-4
  sw $ra,0($sp)
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $v0,0($sp)
  la $a0,promptAlgEncrptB
  jal printStr
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
  move $t2,$zero
nextChrEncryptB:
  lb $t1,($t0)
  beqz $t1,endBufferEncryptB
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $t1,0($sp)
  add $sp,$sp,-4
  sw $t2,0($sp)
  add $sp,$sp,-4
  sw $v0,0($sp)
  move $a0,$t2
  li $a1,2
  jal module                            # Calcolated the module the
                                        # position in base 2
  move $t3,$v0
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t2,0($sp)
  add $sp,$sp,4
  lw $t1,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
  bnez $t3,jumpEncodingB                # Chech if the position is even
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $t2,0($sp)
  add $sp,$sp,-4
  sw $v0,0($sp)
  move $a0,$t1                          # Start - char encoding
  addi $a0,4                            # .
  li $a1,256                            # .
  jal module                            # End - char encoding
  move $t1,$v0
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t2,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
jumpEncodingB:
  sb $t1,0($t0)                         # Store coded char
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $t2,0($sp)
  add $sp,$sp,-4
  sw $v0,0($sp)
  move $a0,$t1
  jal printChr
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t2,0($sp)
  add $sp,$sp,4
```

```
  lw $t0,0($sp)
  add $sp,$sp,4
  add $t2,$t2,1
  add $t0,$t0,1
  add $v0,$v0,1
  j nextChrEncryptB

endBufferEncryptB:
  lw $ra,0($sp)
  addi $sp,$sp,4
  jr $ra

########################################
#                decryptB              #
########################################
decryptB:
  # Procedure for decryption algorithm B
  # $a0: coded message
  # $v0: length of the decrypted message

  move $t0,$a0                          # $t0: Message
  move $v0,$zero
  add $sp,$sp,-4
  sw $ra,0($sp)
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $v0,0($sp)
  la $a0,promptAlgDecrptB
  jal printStr
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
  move $t2,$zero
nextChrDecryptB:
  lb $t1,($t0)
  beqz $t1,endBufferDecryptB
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $t1,0($sp)
  add $sp,$sp,-4
  sw $t2,0($sp)
  add $sp,$sp,-4
  sw $v0,0($sp)
  move $a0,$t2
  li $a1,2
  jal module                            # Calcolated the module the
                                        # position in base 2
  move $t3,$v0
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t2,0($sp)
  add $sp,$sp,4
  lw $t1,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
  bnez $t3,jumpDecodingB                # Chech if the position is even
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $t2,0($sp)
  add $sp,$sp,-4
  sw $v0,0($sp)
  move $a0,$t1                          # Start - char decoding
  addi $a0,-4                           # .
  li $a1,256                            # .
  jal module                            # End - char decoding
  move $t1,$v0
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t2,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
```

```
  add $sp,$sp,4
jumpDecodingB:
  sb $t1,0($t0)                          # Store decoded char
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $t2,0($sp)
  add $sp,$sp,-4
  sw $v0,0($sp)
  move $a0,$t1
  jal printChr
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t2,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
  add $t2,$t2,1
  add $t0,$t0,1
  add $v0,$v0,1
  j nextChrDecryptB

endBufferDecryptB:
  lw $ra,0($sp)
  addi $sp,$sp,4
  jr $ra

########################################
#               encryptC               #
########################################
encryptC:
  # Procedure for algorithm C
  # $a0: original message
  # $v0: length of the encrypted message

  move $t0,$a0                           # $t0: Message
  move $v0,$zero
  add $sp,$sp,-4
  sw $ra,0($sp)
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $v0,0($sp)
  la $a0,promptAlgEncrptC
  jal printStr
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
  move $t2,$zero
nextChrEncryptC:
  lb $t1,($t0)
  beqz $t1,endBufferEncryptC
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $t1,0($sp)
  add $sp,$sp,-4
  sw $t2,0($sp)
  add $sp,$sp,-4
  sw $v0,0($sp)
  move $a0,$t2
  li $a1,2
  jal module                             # Calcolated the module the
                                         # position in base 2
  move $t3,$v0
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t2,0($sp)
  add $sp,$sp,4
  lw $t1,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
  beqz $t3,jumpEncodingC                 # Chech if the position is odd
  add $sp,$sp,-4
```

```
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $t2,0($sp)
  add $sp,$sp,-4
  sw $v0,0($sp)
  move $a0,$t1                          # Start - char encoding
  addi $a0,4                            # .
  li $a1,256                            # .
  jal module                            # End - char encoding
  move $t1,$v0
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t2,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
jumpEncodingC:
  sb $t1,0($t0)                         # Store coded char
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $t2,0($sp)
  add $sp,$sp,-4
  sw $v0,0($sp)
  move $a0,$t1
  jal printChr
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t2,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
  add $t2,$t2,1
  add $t0,$t0,1
  add $v0,$v0,1
  j nextChrEncryptC

endBufferEncryptC:
  lw $ra,0($sp)
  addi $sp,$sp,4
  jr $ra

########################################
#               decryptC               #
########################################
decryptC:
  # Procedure for decryption algorithm B
  # $a0: coded message
  # $v0: length of the decrypted message

  move $t0,$a0                          # $t0: Message
  move $v0,$zero
  add $sp,$sp,-4
  sw $ra,0($sp)
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $v0,0($sp)
  la $a0,promptAlgDecrptC
  jal printStr
  move $t2,$zero
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
  move $t2,$zero
nextChrDecryptC:
  lb $t1,($t0)
  beqz $t1,endBufferDecryptC
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $t1,0($sp)
  add $sp,$sp,-4
  sw $t2,0($sp)
  add $sp,$sp,-4
```

```
  sw $v0,0($sp)
  move $a0,$t2
  li $a1,2
  jal module                          # Calcolated the module the
                                      # position in base 2
  move $t3,$v0
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t2,0($sp)
  add $sp,$sp,4
  lw $t1,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
  beqz $t3,jumpDecodingC              # Chech if the position is odd
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $t2,0($sp)
  add $sp,$sp,-4
  sw $v0,0($sp)
  move $a0,$t1                        # Start - char decoding
  addi $a0,-4                         # .
  li $a1,256                          # .
  jal module                          # End - char decoding
  move $t1,$v0
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t2,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
jumpDecodingC:
  sb $t1,0($t0)                       # Store coded char
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $t2,0($sp)
  add $sp,$sp,-4
  sw $v0,0($sp)
  move $a0,$t1
  jal printChr
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t2,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
  add $t2,$t2,1
  add $t0,$t0,1
  add $v0,$v0,1
  j nextChrDecryptC

endBufferDecryptC:
  lw $ra,0($sp)
  addi $sp,$sp,4
  jr $ra

########################################
#              encryptD                #
########################################
encryptD:
  # Procedure for algorithm D
  # $a0: original message
  # $v0: length of the encrypted message

  move $t0,$a0                        # $t0: Head of message
  move $t1,$a0                        # $t1: Tail of message
  move $t5,$a0                        # $t5: Message
  addi $sp,$sp,-4
  sw $ra,0($sp)
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $t1,0($sp)
  add $sp,$sp,-4
```

```
  sw $t5,0($sp)
  la $a0,promptAlgEncrptD
  jal printStr
  lw $t5,0($sp)
  add $sp,$sp,4
  lw $t1,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $t1,0($sp)
  add $sp,$sp,-4
  sw $t5,0($sp)
  move $a0,$t0
  jal length
  move $t2,$v0                          # $t2: Length of message
  lw $t5,0($sp)
  add $sp,$sp,4
  lw $t1,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
  move $a1,$zero
  addi $a2,$t2,-1
  add $t1,$t1,$a2                       # Moved $t1 at the end of message
nextChrEncryptD:
  lb $t3,($t0)
  lb $t4,($t1)
  sub $a3,$t1,$t0
  blez $a3,endBufferEncryptD            # Check the half of the message
  sb $t4,($t0)                          # Store char in new position
  sb $t3,($t1)                          # Store char in new position
  addi $t0,$t0,1
  addi $t1,$t1,-1
  j nextChrEncryptD

endBufferEncryptD:
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $t1,0($sp)
  add $sp,$sp,-4
  sw $t2,0($sp)
  add $sp,$sp,-4
  sw $t5,0($sp)
  move $a0,$t5
  jal printBuffer
  lw $t5,0($sp)
  add $sp,$sp,4
  lw $t2,0($sp)
  add $sp,$sp,4
  lw $t1,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
  lw $ra,0($sp)
  add $t2,$t2,1
  move $v0,$t2
  addi $sp,$sp,4
  jr $ra

########################################
#                decryptD              #
########################################
decryptD:
  # Procedure for decryption algorithm D
  # $a0: coded message
  # $v0: lenght of the decrypted message

  addi $sp,$sp,-4
  sw $ra,0($sp)
  jal encryptD
  lw $ra,0($sp)
  addi $sp,$sp,4
```

```
  jr $ra

########################################
#                encryptE              #
########################################
encryptE:
  # Procedure for algorithm E
  # $a0: original message
  # $v0: length of the encrypted message

  # Make a copy of original message
  move $t0,$a0                          # $t0: Original message
  la $t1,bufferEncrptDataTmp            # $t1: Temporaty copy
loopCopyEncryptE:
  lb $t2,($t0)
  lb $t3,($t1)
  beqz $t2,endCopyEncryptE
  sb $t2,($t1)
  add $t0,$t0,1
  add $t1,$t1,1
  j loopCopyEncryptE
endCopyEncryptE:

  move $t0,$a0                          # $t0: Original message
  move $t3,$a0                          # $t3: Original message
  add $sp,$sp,-4
  sw $ra,0($sp)
  add $sp,$sp,-4
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $t3,0($sp)
  la $a0,promptAlgEncrptE               # Print pront message
  jal printStr
  lw $t3,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
  move $t8,$zero                        # $t8: Head
  move $t9,$zero                        # $t9: Tail

# Create a list with all the characters present in the message
nextChrEncryptE:
  lb $t2,($t0)
  beqz $t2,endBufferEncryptE
  bne $t8,$zero, linkLast
  addi $sp,$sp,-4                       # Start - Added first char
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $t2,0($sp)
  add $sp,$sp,-4
  sw $t3,0($sp)
  add $sp,$sp,-4
  sw $t8,0($sp)
  add $sp,$sp,-4
  sw $t9,0($sp)
  li $a0,5
  jal sbrk
  lw $t9,0($sp)
  add $sp,$sp,4
  lw $t8,0($sp)
  add $sp,$sp,4
  lw $t3,0($sp)
  add $sp,$sp,4
  lw $t2,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
  sb $t2,4($v0)
  sw $zero,0($v0)
  move $t8,$v0
  move $t9,$v0
  j endOfLoop                           # End - Added first char
linkLast:
  move $t7,$t8                          # $t7 = head of list
loopSearchChar:
  beq $t7,$zero,addNewElement
```

```
  lb $a2,4($t7)
  beq $a2,$t2,endOfLoop
  lw $t7,0($t7)
  j loopSearchChar

addNewElement:
  add $sp,$sp,-4                    # Start - Added new char
  sw $t0,0($sp)
  add $sp,$sp,-4
  sw $t2,0($sp)
  add $sp,$sp,-4
  sw $t3,0($sp)
  add $sp,$sp,-4
  sw $t8,0($sp)
  add $sp,$sp,-4
  sw $t9,0($sp)
  li $a0,5
  jal sbrk
  lw $t9,0($sp)
  add $sp,$sp,4
  lw $t8,0($sp)
  add $sp,$sp,4
  lw $t3,0($sp)
  add $sp,$sp,4
  lw $t2,0($sp)
  add $sp,$sp,4
  lw $t0,0($sp)
  add $sp,$sp,4
  sw $v0,0($t9)
  sb $t2,4($v0)
  sw $zero,0($v0)
  move $t9,$v0                      # End - Added new char

endOfLoop:
  add $t0,$t0,1
  j nextChrEncryptE

endBufferEncryptE:
  move $t7,$t8                      # $t7 = head of list
  move $v0,$zero
loopPrint:
  beq $t7,$zero,endPrint
  lb $a0,4($t7)                     # Store a char
  sb $a0,0($t3)
  add $t3,$t3,1
  addi $v0,$v0,1                    # Increase the length of the message
  add $sp,$sp,-4
  sw $t3,0($sp)
  add $sp,$sp,-4
  sw $t7,0($sp)
  add $sp,$sp,-4
  sw $v0,0($sp)
  lb $a0,4($t7)
  jal printChr                      # Print a char
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t7,0($sp)
  add $sp,$sp,4
  lw $t3,0($sp)
  add $sp,$sp,4
  # Store char
  la $t1,bufferEncrptDataTmp
  lb $a0,positionCounter            # Position counter
  lb $a1,4($t7)                     # Char
loopSearchPosition:
  lb $t2,0($t1)
  beqz $t2,endLooopSearchPosition
  bne $t2,$a1,noPrintPosition
  lb $t4,positionSeparetor          # Store positionSeparetor
  sb $t4,0($t3)
  addi $t3,$t3,1
  addi $v0,$v0,1                    # Increase the length of the message
  move $t4,$a0
  addi $t5,$zero,100
  div $t4,$t5
  mflo $t4
```

```
  beqz $t4, tens
  add $t4,$t4,48
  sb $t4,0($t3)                    # Store position hundreds
  add $t3,$t3,1
  addi $v0,$v0,1                   # Increase the length of the message
  mfhi $t4
  addi $t4,$t4,-10
  bgez $t4, tens                   # Check if remainder of div great or
                                   # equal to zero
  addi $t4,$zero,48
  sb $t4,0($t3)                    # Store position 0 in the tens
  add $t3,$t3,1
  addi $v0,$v0,1                   # Increase the length of the message
tens:
  mfhi $t4
  add $t5,$zero,10
  div $t4,$t5
  mflo $t4
  beqz $t4, units
  addi $t4,$t4,48
  sb $t4,0($t3)                    # Store position tens
  addi $t3,$t3,1
  addi $v0,$v0,1                   # Increase the length of the message
units:
  mfhi $t4
  addi $t4,$t4,48
  sb $t4,0($t3)                    # Store position units
  addi $t3,$t3,1
  addi $v0,$v0,1                   # Increase the length of the message
  addi $sp,$sp,-4
  sw $t7,0($sp)
  addi $sp,$sp,-4
  sw $t1,0($sp)
  addi $sp,$sp,-4
  sw $a0,0($sp)
  addi $sp,$sp,-4
  sw $a1,0($sp)
  addi $sp,$sp,-4
  sw $v0,0($sp)
  la $a0,positionSeparetor         # Print positionSeparator
  jal printStr
  lw $v0,0($sp)
  addi $sp,$sp,4
  lw $a1,0($sp)
  addi $sp,$sp,4
  lw $a0,0($sp)
  addi $sp,$sp,4
  lw $t1,0($sp)
  addi $sp,$sp,4
  lw $t7,0($sp)
  addi $sp,$sp,4
  addi $sp,$sp,-4
  sw $t7,0($sp)
  addi $sp,$sp,-4
  sw $t1,0($sp)
  addi $sp,$sp,-4
  sw $a0,0($sp)
  addi $sp,$sp,-4
  sw $a1,0($sp)
  addi $sp,$sp,-4
  sw $v0,0($sp)
  jal printInt                     # Print position
  lw $v0,0($sp)
  addi $sp,$sp,4
  lw $a1,0($sp)
  addi $sp,$sp,4
  lw $a0,0($sp)
  addi $sp,$sp,4
  lw $t1,0($sp)
  addi $sp,$sp,4
  lw $t7,0($sp)
  addi $sp,$sp,4
noPrintPosition:
  add $t1,$t1,1
  add $a0,$a0,1
  j loopSearchPosition
```

```
endLooopSearchPosition:
  lw $t7,0($t7)
  lb $a0,charSeparetor                    # Store char separator
  sb $a0,0($t3)
  add $t3,$t3,1
  addi $v0,$v0,1                          # Increase the length of the message
  add $sp,$sp,-4
  sw $t7,0($sp)
  add $sp,$sp,-4
  sw $v0,0($sp)
  la $a0,charSeparetor                    # Print char sperator
  jal printStr
  lw $v0,0($sp)
  add $sp,$sp,4
  lw $t7,0($sp)
  add $sp,$sp,4
  j loopPrint
endPrint:

  lw $ra,0($sp)
  addi $sp,$sp,4
  jr $ra

########################################
#                decryptE              #
########################################
decryptE:
  # Procedure for algorithm E
  # $a0: coded message
  # $v0: lenght of the decrypted message

  move $t0,$a0                            # Message
  la $t1,bufferDecrptDataTmp              # Temp buffer
loopCopyDecryptE:                         # Copy the encrypt message in the
  lb $t2,0($t0)                           # temp buffer and clear the original
  beqz $t2,endCopyDecryptE
  sb $t2,0($t1)
  sb $zero,0($t0)
  add $t0,$t0,1
  add $t1,$t1,1
  j loopCopyDecryptE
endCopyDecryptE:
  move $t0,$a0                            # Message
  la $t1,bufferDecrptDataTmp
  move $v0,$zero
  addi $sp,$sp,-4
  sw $ra,0($sp)
  addi $sp,$sp,-4
  sw $t0,0($sp)
  addi $sp,$sp,-4
  sw $t1,0($sp)
  addi $sp,$sp,-4
  sw $v0,0($sp)
  la $a0,promptAlgDecrptE
  jal printStr
  lw $v0,0($sp)
  addi $sp,$sp,4
  lw $t1,0($sp)
  addi $sp,$sp,4
  lw $t0,0($sp)
  addi $sp,$sp,4
nextChrDecryptE:
  lb $t2,0($t1)                           # Char
  beqz $t2,cleanTmpBuffer
  addi $t1,$t1,2
nextPosition:
  lb $t3,0($t1)                           # Position (first digit)
  addi $t3,$t3,-48
nextDigitOfPosition:
  addi $t1,$t1,1
  lb $t4,0($t1)                           # Take the next char
  addi $t4,$t4,-48
  bgez $t4,updatedPosition                # If is a number add digit to position
  add $t0,$t0,$t3                         # In $t3 there is a position of char
  sb $t2,0($t0)                           # Store char
  sub $t0,$t0,$t3
```

```
  addi $v0,$v0,1                        # Increase the length of the message
  addi $t1,$t1,1
  addi $t4,$t4,3
  beqz $t4,nextPosition
  j nextChrDecryptE
updatedPosition:
  mul $t3,$t3,10
  add $t3,$t3,$t4
  j nextDigitOfPosition

cleanTmpBuffer:                         # Clear the temp buffer
  la $t1,bufferDecrptDataTmp
loopCleanTmp:
  lb $t2,0($t1)
  beqz $t2,endBufferDecryptE
  sb $zero,0($t1)
  add $t1,$t1,1
  j loopCleanTmp

endBufferDecryptE:
  addi $sp,$sp,-4
  sw $t0,0($sp)
  addi $sp,$sp,-4
  sw $v0,0($sp)
  move $a0,$t0
  jal printBuffer
  lw $v0,0($sp)
  addi $sp,$sp,4
  lw $t0,($sp)
  addi $sp,$sp,4
  lw $ra,0($sp)
  addi $sp,$sp,4
  jr $ra

########################################
#              encryptMsg              #
########################################
encryptMsg:
  # Procedure to encrypth message
  # This procedure applies the right algorithm or the algorithms on the message
  # in base on the key of the encryption.
  # $a0: encrypth key
  # $a1: length of Key
  # $v0: length of the encrypted message

  # Prepare the jump table of algorithms
  la $t1,jumpEncrtpTable

  move $t3,$a0                      # $t3 key
  move $t5,$a1                      # $t5 length of key
  sub $t5,$t5,1
  addi $sp,$sp,-4
  sw $ra,0($sp)
nextEncrpt:
  lb $t4,($t3)
  beqz $t5,endEncrpt
  sub $t0,$t4,65
  mul $t0,$t0,4
  add $t0,$t0,$t1
  lw $t0,0($t0)
  jr $t0

encrptA:
  addi $sp,$sp,-4
  sw $t0,0($sp)
  addi $sp,$sp,-4
  sw $t1,0($sp)
  addi $sp,$sp,-4
  sw $t3,0($sp)
  addi $sp,$sp,-4
  sw $t4,0($sp)
  addi $sp,$sp,-4
  sw $t5,0($sp)
  la $a0,bufferMsgData
  jal encryptA
  lw $t5,0($sp)
```

```
  addi $sp,$sp,4
  lw $t4,0($sp)
  addi $sp,$sp,4
  lw $t3,0($sp)
  addi $sp,$sp,4
  lw $t1,0($sp)
  addi $sp,$sp,4
  lw $t0,0($sp)
  addi $sp,$sp,4
  j exitCaseEncrpt
encrptB:
  addi $sp,$sp,-4
  sw $t0,0($sp)
  addi $sp,$sp,-4
  sw $t1,0($sp)
  addi $sp,$sp,-4
  sw $t3,0($sp)
  addi $sp,$sp,-4
  sw $t4,0($sp)
  addi $sp,$sp,-4
  sw $t5,0($sp)
  la $a0,bufferMsgData
  jal encryptB
  lw $t5,0($sp)
  addi $sp,$sp,4
  lw $t4,0($sp)
  addi $sp,$sp,4
  lw $t3,0($sp)
  addi $sp,$sp,4
  lw $t1,0($sp)
  addi $sp,$sp,4
  lw $t0,0($sp)
  addi $sp,$sp,4
  j exitCaseEncrpt
encrptC:
  addi $sp,$sp,-4
  sw $t0,0($sp)
  addi $sp,$sp,-4
  sw $t1,0($sp)
  addi $sp,$sp,-4
  sw $t3,0($sp)
  addi $sp,$sp,-4
  sw $t4,0($sp)
  addi $sp,$sp,-4
  sw $t5,0($sp)
  la $a0,bufferMsgData
  jal encryptC
  lw $t5,0($sp)
  addi $sp,$sp,4
  lw $t4,0($sp)
  addi $sp,$sp,4
  lw $t3,0($sp)
  addi $sp,$sp,4
  lw $t1,0($sp)
  addi $sp,$sp,4
  lw $t0,0($sp)
  addi $sp,$sp,4
  j exitCaseEncrpt
encrptD:
  addi $sp,$sp,-4
  sw $t0,0($sp)
  addi $sp,$sp,-4
  sw $t1,0($sp)
  addi $sp,$sp,-4
  sw $t3,0($sp)
  addi $sp,$sp,-4
  sw $t4,0($sp)
  addi $sp,$sp,-4
  sw $t5,0($sp)
  la $a0,bufferMsgData
  jal encryptD
  lw $t5,0($sp)
  addi $sp,$sp,4
  lw $t4,0($sp)
  addi $sp,$sp,4
  lw $t3,0($sp)
```

```
  addi $sp,$sp,4
  lw $t1,0($sp)
  addi $sp,$sp,4
  lw $t0,0($sp)
  addi $sp,$sp,4
  j exitCaseEncrpt
encrptE:
  addi $sp,$sp,-4
  sw $t0,0($sp)
  addi $sp,$sp,-4
  sw $t1,0($sp)
  addi $sp,$sp,-4
  sw $t3,0($sp)
  addi $sp,$sp,-4
  sw $t4,0($sp)
  addi $sp,$sp,-4
  sw $t5,0($sp)
  la $a0,bufferMsgData
  jal encryptE
  lw $t5,0($sp)
  addi $sp,$sp,4
  lw $t4,0($sp)
  addi $sp,$sp,4
  lw $t3,0($sp)
  addi $sp,$sp,4
  lw $t1,0($sp)
  addi $sp,$sp,4
  lw $t0,0($sp)
  addi $sp,$sp,4
exitCaseEncrpt:
  sub $t5,$t5,1
  add $t3,$t3,1
  j nextEncrpt

endEncrpt:
  lw $ra,0($sp)
  addi $sp,$sp,4
  jr $ra

#########################################
#              decryptMsg               #
#########################################
decryptMsg:
  # Procedure to decrypth the encrypted message
  # This procedure applies the right algorithm or the algorithms on the message
  # in base on the key of the encryption.
  # $a0: Encrypth key
  # $a1: length of Key
  # $v0: length of the decrypted message

  # Prepare the jump table of algorithms
  la $t1,jumpDecrptTable

  move $t3,$a0 #                $t3 key
  move $t5,$a1 #                $t5 length of key
  sub $t5,$t5,2
  add $t3,$t3,$t5
  addi $sp,$sp,-4
  sw $ra,0($sp)
nextDecrpt:
  lb $t4,($t3)
  bltz $t5,endDecrpt
  sub $t0,$t4,65
  mul $t0,$t0,4
  add $t0,$t0,$t1
  lw $t0,0($t0)
  jr $t0

decrptA:
  addi $sp,$sp,-4
  sw $t0,0($sp)
  addi $sp,$sp,-4
  sw $t1,0($sp)
  addi $sp,$sp,-4
  sw $t3,0($sp)
  addi $sp,$sp,-4
```

```
  sw $t4,0($sp)
  addi $sp,$sp,-4
  sw $t5,0($sp)
  la $a0, bufferEncrptData
  jal decryptA
  lw $t5,0($sp)
  addi $sp,$sp,4
  lw $t4,0($sp)
  addi $sp,$sp,4
  lw $t3,0($sp)
  addi $sp,$sp,4
  lw $t1,0($sp)
  addi $sp,$sp,4
  lw $t0,0($sp)
  addi $sp,$sp,4
  j exitCaseDecrpt
decrptB:
  addi $sp,$sp,-4
  sw $t0,0($sp)
  addi $sp,$sp,-4
  sw $t1,0($sp)
  addi $sp,$sp,-4
  sw $t3,0($sp)
  addi $sp,$sp,-4
  sw $t4,0($sp)
  addi $sp,$sp,-4
  sw $t5,0($sp)
  la $a0, bufferEncrptData
  jal decryptB
  lw $t5,0($sp)
  addi $sp,$sp,4
  lw $t4,0($sp)
  addi $sp,$sp,4
  lw $t3,0($sp)
  addi $sp,$sp,4
  lw $t1,0($sp)
  addi $sp,$sp,4
  lw $t0,0($sp)
  addi $sp,$sp,4
  j exitCaseDecrpt
decrptC:
  addi $sp,$sp,-4
  sw $t0,0($sp)
  addi $sp,$sp,-4
  sw $t1,0($sp)
  addi $sp,$sp,-4
  sw $t3,0($sp)
  addi $sp,$sp,-4
  sw $t4,0($sp)
  addi $sp,$sp,-4
  sw $t5,0($sp)
  la $a0, bufferEncrptData
  jal decryptC
  lw $t5,0($sp)
  addi $sp,$sp,4
  lw $t4,0($sp)
  addi $sp,$sp,4
  lw $t3,0($sp)
  addi $sp,$sp,4
  lw $t1,0($sp)
  addi $sp,$sp,4
  lw $t0,0($sp)
  addi $sp,$sp,4
  j exitCaseDecrpt
decrptD:
  addi $sp,$sp,-4
  sw $t0,0($sp)
  addi $sp,$sp,-4
  sw $t1,0($sp)
  addi $sp,$sp,-4
  sw $t3,0($sp)
  addi $sp,$sp,-4
  sw $t4,0($sp)
  addi $sp,$sp,-4
  sw $t5,0($sp)
  la $a0, bufferEncrptData
```

```
  jal decryptD
  lw $t5,0($sp)
  addi $sp,$sp,4
  lw $t4,0($sp)
  addi $sp,$sp,4
  lw $t3,0($sp)
  addi $sp,$sp,4
  lw $t1,0($sp)
  addi $sp,$sp,4
  lw $t0,0($sp)
  addi $sp,$sp,4
  j exitCaseDecrpt
decrptE:
  addi $sp,$sp,-4
  sw $t0,0($sp)
  addi $sp,$sp,-4
  sw $t1,0($sp)
  addi $sp,$sp,-4
  sw $t3,0($sp)
  addi $sp,$sp,-4
  sw $t4,0($sp)
  addi $sp,$sp,-4
  sw $t5,0($sp)
  la $a0,bufferEncrptData
  jal decryptE
  lw $t5,0($sp)
  addi $sp,$sp,4
  lw $t4,0($sp)
  addi $sp,$sp,4
  lw $t3,0($sp)
  addi $sp,$sp,4
  lw $t1,0($sp)
  addi $sp,$sp,4
  lw $t0,0($sp)
  addi $sp,$sp,4
exitCaseDecrpt:
  sub $t5,$t5,1
  sub $t3,$t3,1
  j nextDecrpt

endDecrpt:
  lw $ra,0($sp)
  addi $sp,$sp,4
  jr $ra

#################################### MAIN ####################################
main:
##############################################################################

  addi $sp,$sp,-4
  sw $s0,0($sp)
  addi $sp,$sp,-4
  sw $s1,0($sp)
  addi $sp,$sp,-4
  sw $s2,0($sp)
  addi $sp,$sp,-4
  sw $s3,0($sp)
  addi $sp,$sp,-4
  sw $s4,0($sp)
  addi $sp,$sp,-4
  sw $s5,0($sp)
  addi $sp,$sp,-4
  sw $s6,0($sp)
  addi $sp,$sp,-4
  sw $s7,0($sp)

  # Open file to read the key
  la $a0,promptReadKey
  jal printStr
  la $a0,fullNameOfKey
  jal openFileToRead

  # Check if file of key exist
  beq $v0,-1,keyNotExist

  # Read the key
```

```
move $a0,$v0
la $a1,bufferKeyData
li $a2,5
jal readFile
move $s2,$v0

# Check if file of key is empty
beq $s2,$zero,keyIsEmpty

# Open file to read message
la $a0,promptReadMsg
jal printStr
la $a0,fullNameOfMsg
jal openFileToRead

# Check if file of message exist
beq $v0,-1,msgNotExist

# Read the message
move $a0,$v0
la $a1,bufferMsgData
li $a2,128
jal readFile
move $s3,$v0

# Check if file of key is empty
beq $s3,$zero,msgIsEmpty

# Diplay the length of message
la $a0,promptLengthMsg
jal printStr
move $a0,$s3
jal printInt
la $a0,lineFeed
jal printStr

###############################################################################
#                                 Encrypt message                             #
###############################################################################
la $a0,promptEncrptMsg
jal printStr
la $a0,bufferKeyData
move $a1,$s2
jal encryptMsg
move $s3,$v0

# Open file to write the encrypth message
la $a0,promptWriteEncrptMsg
jal printStr
la $a0,fullNameOfEncrptMsg
jal openFileToWrite
move $s4,$v0

# Write the encrypted message
move $a0,$s4
la $a1,bufferMsgData
move $a2,$s3
jal writeFile

# Close the encrypted message
move $a0,$s4
jal closeFile

###############################################################################
#                                 Decrypt message                             #
###############################################################################
# Open file to read coded message
la $a0,fullNameOfEncrptMsg
jal openFileToRead

# Read the encrypth message
move $a0,$v0
la $a1,bufferEncrptData
move $a2,$s3
jal readFile
move $s3,$v0
```

```
  la $a0, promptDecrptMsg
  jal printStr
  la $a0, bufferKeyData
  move $a1, $s2
  jal decryptMsg
  move $s3, $v0

  # Open file to write the decrypted message
  la $a0, promptWriteDecrptMsg
  jal printStr
  la $a0, fullNameOfDecrptMsg
  jal openFileToWrite
  move $s4, $v0

  # Write the decrypted message
  move $a0, $s4
  la $a1, bufferEncrptData
  move $a2, $s3
  jal writeFile

  # Close the decrypted message
  move $a0, $s4
  jal closeFile

  j endProgram

keyNotExist:
  la $a0, msgErrKeyNotExist
  j printErrMsg

keyIsEmpty:
  la $a0, msgErrKeyIsEmpty
  j printErrMsg

msgNotExist:
  la $a0, msgErrMsgNotExist
  j printErrMsg

msgIsEmpty:
  la $a0, msgErrMsgIsEmpty
  j printErrMsg

printErrMsg:
  jal printStr

endProgram: # Exit to programm
  lw $s7, 0($sp)
  addi $sp, $sp, 4
  lw $s6, 0($sp)
  addi $sp, $sp, 4
  lw $s5, 0($sp)
  addi $sp, $sp, 4
  lw $s4, 0($sp)
  addi $sp, $sp, 4
  lw $s3, 0($sp)
  addi $sp, $sp, 4
  lw $s2, 0($sp)
  addi $sp, $sp, 4
  lw $s1, 0($sp)
  addi $sp, $sp, 4
  lw $s0, 0($sp)
  addi $sp, $sp, 4
  la $a0, msgEndProgram
  jal printStr
  li $v0, 10
  syscall
```