PROYECTO 7

Grupo 11: Reyes Del Viejo Gomez, Ignacio
Martínez Ruiz, Noemí Villarroya Marco

Descripción del funcionamiento del programa principal:

El programa consiste en pedir al usuario una cadena (hasta 100 caracteres) la cual vamos a encriptar o desencriptar a través de una semilla implementada (número entre 0 y 31). También nos dan un vector de 31 números enteros elegidos de forma aleatoria entre -15 y 31, donde gracias a la semilla podremos posicionarnos en dicho vector. Además necesitaremos una letra “e” (mayúscula o minúscula) para encriptar y otra letra “d” (también en mayúscula o en minúscula) para desencriptar.

Para encriptar: Con la posición de la semilla localizada en el vector, mediante una suma entre el valor del primer carácter de la cadena en ASCII y el valor de la posición semilla en el vector, obtendremos el valor del carácter encriptado, a partir de aquí, se cogerá el valor del siguiente carácter en ASCII y se le sumará 1 a la semilla y se repetirá este último proceso hasta que recorramos toda la cadena.

Para desencriptar: Una vez recorrida toda la cadena, tendremos que revertir el texto encriptado, lo que haremos será coger el valor del vector al que apunte la semilla y le restamos el contenido al carácter encriptado. Luego sumamos uno a la semilla, y repetimos un proceso similar al de encriptado, con la diferencia que habrá que restar en vez de sumar.

Cada carácter a encriptar, si es una letra, se pasará a mayúscula. Los espacios no se van a encriptar, quedando como espacios en el mensaje encriptado. Si se encuentra un carácter que no sea una letra, una cifra decimal o un espacio, se mostrará un mensaje en el display LCD indicando cuál ha sido el carácter encontrado y qué posición ocupa en la cadena de caracteres.

El programa no debe aceptar cadenas de texto de longitud nula ni superior a 100 caracteres ni que contengan caracteres que no sean letras ni cifras decimales. Tampoco aceptará valores de la semilla no comprendidos entre 0 y 31 ni letras de acción (encriptar – desencriptar) diferentes de “e” o “d”. En cualquiera de estos casos, el programa finalizará mostrando el motivo en el display LCD con un mensaje específico.

Descripción de las subrutinas a usar:
Habrá una rutina encriptar, otra desencriptar.

Parámetros de entrada de cada subrutina:
encriptar:
carácter actual de la cadena (valor). Registro r1.
vector (referencia). Registro r3
semilla (valor). Registro r2

desencriptar:
carácter actual de la cadena (valor). Registro r1.
vector (referencia). Registro r3
semilla (valor). Registro r2

Parámetros de salida de cada subrutina:
encriptar:
carácter actual de la cadena encriptada (valor). Registro r2

desencriptar:
carácter actual de la cadena desencriptada (valor). Registro r2

Estructura y descripción de cada bloque de activación que se emplea:
En total tenemos 8 bloques de activación: 2 while y 6 if.

Un if lo usamos al principio para asegurar que el dato que viene en la etiqueta “EoD” de .data es una “E” de encriptar o una “D” desencriptar (ambas mayúsculas).

Si la letra en r4 es una “E” nos iremos a la etiqueta “Enc”, de encriptar, y si la letra es una “D” nos iremos a la etiqueta “Desnc”, de desencriptar. Si por un casual, nos introducen algo que no es ni una E ni una D (excepción) , se realiza un salto incondicional a la etiqueta “error” donde se mostrará por pantalla el error y se volverá al programa principal mediante “mov pc, lr”.

```
        mov r6, #0xDF
        and r4, r6 @Para poner la letra que haya en rl en mayúscula
        cmp r4, #'E' @Comparamos si es una E para encriptar
        beq Enc
        cmp r4, #'D' @Comparamos si es una D para desencriptar
        beq Desnc
        b error
        b fin
```

```
error:   @cuando no nos dan la letra E ni D
        @ escribir un mensaje de error en pantalla
         mov pc, lr
```

Luego empezamos a usar los whiles, que están en el programa para encriptar o desencriptar, dependiendo de la letra que haya en la etiqueta “EoD”.

Si la letra es una “E”, nos vamos a ir a la etiqueta “Enc”. Esta etiqueta nos lleva a un while que se va a encargar de recorrer toda la cadena que queremos encriptar, de tal forma que repetimos el while hasta que el carácter actual de la cadena sea igual a 0 (ya que la cadena está guardada en .asciz y cuando esta cadena acaba se guarda un 0).

Dentro de este while, nos encontramos primero con un if/else donde nos aseguramos de que la letra actual en la que estamos es una letra mayúscula que se encuentra entre la “A” y la “Z” (volvemos a hacer una máscara con r6 y esta vez con r1 ya que es el registro donde guardamos el carácter actual de la cadena).

Por tanto, “if” letra actual es menor que ‘A’ o mayor que ‘Z’, vamos a realizar un salto a la etiqueta “noletra” donde mostraremos por pantalla la letra actual en la que estamos y la posición donde nos encontramos en la cadena.

Si el if anterior no se cumple, eso quiere decir que nuestra letra actual es una letra y por tanto vamos a llamar a la subrutina “encriptar”. En este mismo else, tenemos que modificar el valor de la semilla sumándole 1. El problema es que la semilla sólo puede llegar al valor 31 ya que el vector tiene 32 posiciones, y para esto hacemos otro if/else: si la semilla vale 31, modificamos su valor por 0 (se ha “sumado 1”); si no, la semilla no ha llegado todavía a su límite y podemos sumarle 1 tranquilamente.

Este sería el planteamiento en python:

```
Enc:
        @hacemos un while para recorrer toda la cadena que queremos encriptar
        @El problema en python sería:
        @
        @ semilla = 6
        @ vector = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]
        @ cadena = "Texto de prueba: cuantas letras distintas hay?"
        @ posición = 0
        @ caracteractual= direcciónцadena[posición]
        @ cadSolución= []
        @
        @ while caracteractual != 0:
        @
        @       if caracteractual < "A" and  caracteractual > "Z":          ( el caracter actual no es una letra)
        @               print ( caracteractual + posición )
        @
        @
        @       else:     (si es una letra empezamos a encriptar)
        @
        @               letraencriptada= encriptar(semilla, vector, caracteractual)
        @               cadsolución += letraencriptada
        @
        @               if semilla = 31:
        @                       semilla = 0
        @
        @               else:
        @                       semilla += 1           (para que pase al siguiente contenido del vector)
        @
        @       posición += 1
        @       caracteractual= cadena[posición]        (cogemos el siguiente caracter de la cadena)
```

Y este en ensamblador:

```
Enc:
        ldrb r1, [r0, r5]     @el caracter actual de la cadena se guarda en r1
while1: cmp r1, #0       @lo comparamos con cero ya que la cadena está guardada en ascii y esp quiere decir que cuando lleguemos a el caracter 0 se habrá acabado la cadena
        beq finWhile1
        and r1, r6     @r6 = 0xDF
if1:    cmp r1, #'A'
        blt noletra
        cmp r1, #'Z'
        bgt noletra
        b else1
noletra:
        @imprimimos en el simulador de pantalla la posición (r5) y imprimimos el caracter actual (r1)
        b finIE
else1:
        push{r0,r2}
        bl encriptar
        pop{r0}
        ldrb r7, [r2]    @guardamos la letra encriptada en r7
        @tenemos que guardar la letra encriptada en un word o en una cadena que será la cadena solución
        pop{r2}

if2:    cmp r2, #31   @la semilla solo puede llegar hasta 31
        beq limite
        b else2
limite: mov r2, #0
        b FinIE2

else2:  add r2, #1

finIE2:
finIE:
        add r5, #1
        ldr r1, [r0, r5]
        b while1
```

Si la letra de la etiqueta “EoD” era una “D” , vamos a ir a la etiqueta “Desnc”. En esta etiqueta volvemos a tener un while que va a recorrer la cadena que queremos desencriptar (vamos a suponer que la cadena está bien encriptada y que no se encriptaron, ni espacios, ni números decimales, ni nada distinto a una letra).
Antes de empezar este while ,que va a tener la misma condición que el while que usamos para encriptar, guardamos el r1 el primer carácter de la cadena.

Después de desencriptar el carácter de la cadena, modificamos el valor de la semilla tal y como hicimos cuando encriptamos, y luego sumamos a la posición de la cadena un 1 y ponemos como carácter actual al siguiente carácter que hay en la cadena.

Este sería el planteamiento en python:

```
@hacemos un while para recorrer toda la cadena que queremos desencriptar
@El problema en python sería:
@
@ semilla = 6
@ vector = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]
@ cadena = "Texto de prueba: cuantas letras distintas hay?"
@ posición = 0
@ caracteractual= direcciónca dena[posición]
@ cadSolución= []
@
@ while caracteractual != 0:
@
@       letradesencriptada= desencriptar(semilla, vector, caracteractual)
@       cadsolución += letradesencriptada
@
@       if semilla = 31:
@               semilla = 0
@
@       else:
@               semilla += 1           (para que pase al siguiente contenido del vector)
@
@       posición += 1
@       caracteractual= cadena[posición]        (cogemos el siguiente caracter de la cadena)
```

Y este en ensamblador:

```
Desnc:
        ldrb r1, [r0, r5]     @el caracter actual de la cadena se guarda en r1
while2: cmp r1, #0       @lo comparamos con cero ya que la cadena está guardada en ascii y esp quiere decir que cuando lleguemos a el caracter 0 se habrá acabado la cadena
        beq finWhile2
        and r1, r6     @r6 = 0xDF

        push{r0,r2}
        bl desencriptar
        pop{r0}
        ldrb r7, [r2]    @guardamos la letra encriptada en r7
        @tenemos que guardar la letra encriptada en un word o en una cadena que será la cadena solución
        pop{r2}

if3:    cmp r2, #31   @la semilla solo puede llegar hasta 31
        beq limite2
        b else3
limite2:        mov r2, #0
        b FinIE3

else3:  add r2, #1

finIE3:
        add r5, #1
        ldr r1, [r0, r5]
        b while2


finWhile2:      b seguir
```

Otros detalles importantes o significativos:

uso de cada registro (programa principal y en las subrutinas):

-Programa principal:

r0= dirección de la cadena
r1= caracter actual de la cadena
r2= semilla
r3= dirección del vector
r4= variable "E" o "D"
r5= posición
r6= máscara 0xDF
r7=letra encriptada/desencriptada

-Subrutina “encriptar”:

r0= dirección de la cadena
r1= letra actual de la cadena
r2= semilla que varía constantemente
r3= dirección del vector

-Subrutina “desencriptar”:

r0= dirección de la cadena
r1= letra actual de la cadena
r2= semilla
r3= dirección del vector

Tanto en la subrutina “encriptar” como “desencriptar”, tenemos que apilar el contenido de r0 y r2 en la pila con push antes de llamar a las subrutinas porque vamos a modificar estos valores dentro de ellas (r0 guardará ahora el contenido del vector y r2 guardará la letra encriptada o desencriptada). Después de llamar a la subrutina volvemos donde nos quedamos en el programa principal, ponemos un pop{r0} porque es lo que está encima de la pila (la pila va de registro menor a mayor), luego guardamos el contenido de r2 y volvemos a hacer un pop{r2} para devolverle al registro r2 el valor de la semilla.

Destacamos la declaración de varias variables que son necesarias para el programa, empezando por la primera variable semilla la cual empieza en un número y gracias al programa vamos a ir modificando su valor para determinar la posición a qué posición queremos acceder dentro del vector. Otra de las más importantes será la de posición, que actuará como un contador a la hora de recorrer la cadena y además nos ayuda a completar la nueva cadena con los datos encriptados/desencriptados.

En cuanto a las estrategias de acceso a estructuras de datos usamos el “mov pc lr” cuando estamos al final de la subrutina y queremos volver al programa principal. No hemos tenido que guardar en ningún momento el registro lr en la pila ya que no hemos usado subrutinas dentro de subrutinas (no es necesario tener que usar la instrucción “pop {pc}”.