Primer tarea programada, Programa de Cifrado César (Mayo 2011)

Enmanuel O. Ramírez, Yendry R. Rodríguez, Alonso V. Brenes

***Abstract*— El "cifrado de César" o también conocido como desplazamiento de Cesar permitió a Julio César proteger sus mensajes importantes de las miradas no autorizadas, este consiste en substituir cada letra del mensaje por otra que se encuentre un número fijo de posiciones más adelante en el alfabeto; este proyecto está basado en esta técnica de criptografía pero con una implementación en lenguaje ensamblador para x86 con sintaxis AT&T.**

Será ejecutado en el sistema GNU/Linux donde se presentará la opción de utilizarlo con dos tipos de banderas: *-e* y un valor de 0 a 99 iniciara el proceso de encripción y con *–d* y un valor de 0 a 99 iniciara el proceso de desencripción. Los datos que se codificarán se deben recibir en un archivo con el texto a encriptar en formato ASCII estándar.

***Términos indexados*— César, cifrado, cifrar, criptografía, programa.**

I. Descripción del Problema

En el año 500 a.C. los griegos utilizaron un cilindro llamado "scytale" alrededor del cual enrollaban una tira de cuero. Al escribir un mensaje sobre el cuero y desenrollarlo se veía una lista de letras sin sentido. El mensaje correcto sólo podía leerse al enrollar el cuero nuevamente en un cilindro de igual diámetro.

Durante el Imperio Romano Julio Cesar empleo un sistema de cifrado que consistía en sustituir la letra a encriptar por otra letra distanciada a tres posiciones más adelante. Durante su reinado, los mensajes de Julio Cesar nunca fueron desencriptados.[1]

Este proyecto consiste en la implementación de un programa de criptología, basado en el método de cifrado de César también conocido como cifra modificada de César, el método se basa en la sustitución de caracteres, según la clave que s e use el resultado codificado puede variar. El programa debe recibir la fuente del archivo o documento que se quiere encriptar en formato ASCII estándar, un nombre, que será asignado al archivo codificado, una bandera que especifique la operación que se quiere realizar, esta debe ser *–e* para encriptar y *–d* para desencriptar el documento, seguido de la clave con la que va a ser o fue encriptado el archivo. Originalmente se deben utilizar 35 caracteres de la tabla ASCII para encriptar el mensaje, estos son las letras minúsculas del alfabeto de la *a* a la *z* y números del *0* al *9*. Además se debe incluir la opción de utilizar más símbolos que estos.

Para dar solución a este problema, se debe conocer principalmente, como manejar archivos, funciones y buffers, además de conocer acerca de los diferentes tipos de direccionamientos y operaciones básicas en lenguaje ensamblador x86 con sintaxis AT&T.

II. Estrategia de Solución Implementada

Para implementar un programa que cumpliera con los requisitos solicitados, utilizamos principalmente como base el código fuente en C proveído con la descripción del proyecto, sólo con algunas variaciones y funciones propias.

Primeramente se seleccionó una serie de caracteres de la tabla ASCII para que fueran los utilizables en el método de encripción y desencripción. Entre estos se encuentran los caracteres imprimibles y otros de control y estructura. Con este grupo se cubren los símbolos utilizados en el lenguaje de C.

Luego se generaron métodos para obtener posiciones lineales respecto a la tabla (matriz) a partir de un carácter y viceversa. A partir de ese método, se implementaron otros dos que fueran capaces de cifrar y descifrar un solo carácter a partir de la tabla ya definida, un valor de posición lineal, el cual se puede obtener de las funciones anteriores, y un valor de desplazamiento, utilizado para calcular el nuevo carácter, ya sea encriptado o desencriptado.

Ahora que es posible cifrar y descifrar a nivel de carácter, creamos métodos para realizar lo mismo pero tomando todo el texto de un archivo de entrada y transformándolo. Básicamente se leen trozos de texto que se almacenan en un buffer, se encripta o desencripta y se escribe en el archivo de salida. Este procedimiento se repite hasta que ya no quede texto por leer del archivo original.

Adicionalmente el programa se detiene si encuentra algún error, ya sea en los archivos de entrada o salida, o en los

16 de mayo, 2011.

Yendry Rojas Rodríguez, carné 201025588, estudiante de Ingeniería en Computación, TEC Costa Rica, Sede San Carlos. (email: yenrr16@gmail.com).

Enmanuel Oviedo Ramírez, carné 201041992, estudiante de Ingeniería en Computación, TEC Costa Rica, Sede San Carlos. (email: eoviedo1691@gmail.com).

Luis Alonso Vega Brenes, carné 201042592, estudiante de Ingeniería en Computación, TEC Costa Rica, Sede San Carlos. (email: lavb91@gmail.com).

parámetros. Cuando esto sucede, se muestra una línea con problema encontrado junto con el texto de ayuda del programa, el cual informa al usuario la forma en que se debe utilizar, es decir, el formato de parámetros de entrada del mismo.

III. Conjunto de pruebas

Tabla 1: Ejemplo de cifrado de texto.

Desplazamiento	**Texto codificado**
Texto inicial	`I had seen little of Holmes lately.`
1	`eWr,JW}TTKW7|##7TWU^W[U7AT}W7,#T7V`
2	`oa|6Ta#^^UaA"--A^a_hae_AK^#aA6-^A`&`
3	`yk"@^k-hh_kK,77KhkirkoiKUh-kK@7hKj0`
6	`3%@^|%K""}%iJUUi"%#,%)#is"K%i^U"i$N`
7	`=/Jh"/U,,#/sT__s,/-6/3-s},U/sh_,s.X`
8	`G9Tr,9_66-9}^ii}697@9=7}#6_9}ri6}8b`
9	`QC^|6Ci@@7C#hss#@CAJCGA#-@iC#|s@#Bl`
10	`\Ni#ANtKKBN.s~~.KNLUNRL.8KtN.#~K.Cw`
20	`]Oj$BOkLL9O%tuu%LOCVOSC%/LkO%$uL%Dx`
30	`^Pa 9PlCC:P&kvv&CPDMPTD&0ClP& vC&Ey`
40	`_Qb :QmDD;Q'lww'DQENQUE'1DmQ' wD'Fz`
60	`WSd<SoFF=S)nyy)FSGPSMG)3FoS)yF)H|`
71	`b^o)G^zQQH^4y 4Q^R[^XR4>Qz^4) Q4S#`
80	`YUf >UqHH?U+p{{+HUIRUOI+5HqU+{H+J~`
90	`ZVg!?VrII@V,q||,IVJSVPJ,6IrV,!|I,Ku`
94	`~+Ig~6qqh~T5@@Tq~r{~xrT^q6~TI@qTs9`
97	`<8Ig!8T++"8rS^^r+8,582,r|+T8rg^+r-W`
99	`PL]{5Lh??6L"grr"?L@ILF@",?hL"{r?"Ak`

El texto anterior se cifró con valores de desplazamiento distintos. Luego se utilizó el mismo programa para descifrar los textos de salida utilizando el código de desplazamiento que se usó para cifrarlo y el resultado fue el mismo texto de entrada original (el que fue cifrado al inicio).

Se utilizaron varios textos con diferentes caracteres, para probar que el programa pudiese cifrar y descifrar los textos y en todos los casos el resultado fue el esperado.

IV. Conclusiones

- Con respecto al método Cifrado de César, dio a trabajar en un proyecto interesante en donde aplicamos y reforzamos nuestros conocimientos en el lenguaje ASM x86, además de obtener algunos conocimientos básicos sobre criptografía y un método simple de cifrado.

- Observamos claramente que al resolver un problema en ensamblador, la longitud del programa aumenta considerablemente e incluso se puede volver más complicado de implementar. Por otro lado, el código es ejecutado a menor nivel, por lo que su eficiencia y el control que el programador que tiene sobre él, es mucho mayor que el que se obtiene en lenguajes de alto nivel, lo que compensa la complejidad de implementarlo.

La criptografía es una ciencia interesante que fue aplicada por nuestros antepasados desde el inicio de los tiempos para proteger información confidencial. Con el avance de los años esta técnica ha ido mejorando, siendo cada vez más segura y difícil de romper, Alan Turing, durante la Segunda Guerra Mundial, trabajó en romper los códigos nazis, particularmente los de la máquina Enigma, fue a partir de este momento que la metodología de encriptación enfocada a la computación fue tomando auge. Hoy podemos ver la amplia cantidad de métodos que existen para la protección de datos, un ejemplo es la criptografía cuántica, que utiliza principios de la mecánica cuántica para garantizar la absoluta confidencialidad de la información transmitida. Las actuales técnicas de la criptografía cuántica permiten a dos personas crear, de forma segura, una clave secreta compartida que puede ser usada como llave para cifrar y descifrar mensajes usando métodos de criptografía simétrica.

La criptografía cuántica está cercana a una fase de producción masiva, utilizando láseres para emitir información en el elemento constituyente de la luz, el fotón, y conduciendo esta información a través de fibras ópticas.[2]

V. Referencias bibliográficas

1. Criptología. Cristian Borghello. http://www.segu-info.com.ar/criptologia/criptologia.htm

2. Criptografía cuántica. Wikipedia. http://es.wikipedia.org/wiki/Criptograf%C3%ADa_cu%C3%A1ntica

3. COPIA VERBATIM DEL CÓDIGO FUENTE

```
###############################################################################
#                                                                             #
#                                                                             #
#  cc-kryptos - Programa de encripcion y desencripcion de archivos            #
#                                                                             #
#  Disenado por:                                                              #
#       - Yendry Rojas Rodriguez  (201025588)                                 #
#       - Enmanuel Oviedo Ramírez (201041992)                                 #
#       - Alonso Vega Brenes      (201042592)                                 #
#                                                                             #
#  TEC, Santa Clara                                                           #
#  Arquitectura de Computadores (IC3101)                                      #
#  Profesor Santiago Nunez Corrales                                           #
#  ___________________________________________________________________________#
#                                                                             #
#  Uso:  ./cc-kryptos fin fout flag val                                       #
#                                                                             #
#  fin:  Nombre de archivo de entrada                                         #
#  fout: Nombre de archivo de salida                                          #
#  flag: Bandera de encripcion '-e' o desencripcion '-d'                      #
#  val:  Valor de desplazamiento o codigo de (des)encripcion                  #
#                                                                             #
#                                                                             #
###############################################################################

.section .data

  # Constantes de tamano de columna y fin de linea
  .equ BLOQUE1, 5                   # Tamano del bloque 1 de caracteres
  .equ DIM, 10                      # Dimension de la tabla de caracteres
  .equ TBUFFER, 10                  # Tamano del buffer

  # Constantes de posicion de parametros en funciones
  .equ ARG1, 8                      # Posicion del argumento 1
  .equ ARG2, 12                     # Posicion del argumento 2
  .equ ARG3, 16                     # Posicion del argumento 3
  .equ ARG4, 20                     # Posicion del argumento 4
  .equ ARG5, 24                     # Posicion del argumento 5

  .equ SYSCALL, 0x80                # Codigo de llamada al sistema

  # Manejo de archivos
  .equ READ, 3                      # Leer
  .equ WRITE, 4                     # Escribir
  .equ OPEN, 5                      # Abrir
  .equ CLOSE, 6                     # Cerrar

  # File Descriptors de Consola
  .equ STDIN, 0                     # Lectura de consola
  .equ STDOUT, 1                    # Escritura de consola

  # Opciones de apertura de archivos
  .equ READ_ONLY, 0                 # Solo lectura
  .equ CREAT_WR_TRUNC, 03101        # Crear si no existe, escritura,
                                    # truncar si existe
```

```
  .equ DEFAULT_PER, 0666            # Permiso por defecto

  # Mensajes de error del programa
  ERR_PARAMETROS:
      .ascii   "La cantidad de parametros no es correcta\n\0"
  ERR_ARCHIVO:
      .ascii   "No se pudo abrir alguno de los archivos\n\0"
  ERR_BANDERA:
      .ascii   "Error en la bandera de operacion\n\0"
  ERR_DESPL:
      .ascii   "Error en el codigo de desplazamiento\n\0"

  # Texto de ayuda del programa
  TEXTO_AYUDA:
      .ascii "\tEncripcion de cc-kryptos\n"
      .ascii "\t- - - - - - - - - - - - - - - - - - - - - - -\n"
      .ascii "\t./cc-kryptos fin fout flag val\n"
      .ascii "\tfin: \tnombre de archivo de entrada\n"
      .ascii "\tfout:\tnombre de archivo de salida\n"
      .ascii "\tflag:\tbandera de operacion '-e' o '-d'\n"
      .ascii "\t\t-e: encriptar\n"
      .ascii "\t\t-d: desencriptar\n"
      .ascii "\tval: \t valor de desplazamiento entre 0 y 99\n\0"


CHARS:                              # Vector global de caracteres
  .byte      9, 10, 11, 12, 13      # Caracteres no imprimibles del 0 - 4
  .ascii    " !\"#&%'()*+´-."       # Caracteres del  5 - 19
  .ascii    "/0123456789:;<=>?@AB"  # Caracteres del 20 - 39
  .ascii    "CDEFGHIJKLMNOPQRSTUV"  # Caracteres del 40 - 59
  .ascii    "WXYZ[\\]^_`abcdefghij" # Caracteres del 60 - 79
  .ascii    "klmnopqrstuvwxyz{|}~"  # Caracteres del 80 - 99

CHARS_FIN:                          # Tamano lineal de la tabla
  .equ CHARS_TAM, CHARS_FIN - CHARS

.section .bss
  # Buffer de lectura y escritura para los archivos
  .lcomm BUFFER, TBUFFER            # Tamano definido arriba en TBUFFER

.section .text

  .global _start

  # Argumentos de entrada del programa
  .equ    ARGC, 0                   # Cuenta de argumentos
  .equ    N_PROGRAM, 4              # Nombre del programa
  .equ    A_ENTRADA, 8              # Archivo de entrada
  .equ    A_SALIDA, 12              # Archivo de salida
  .equ    BANDERA, 16               # Bandera -e o -d
  .equ    DESPL, 20                 # Desplazamiento en texto
  # Posicion de los FD de los archivos de entrada y salida
  .equ A_ENTRADA_FD, -4             # File descriptor de entrada
  .equ A_SALIDA_FD, -8              # File descriptor de salida
  .equ ENT_DESPL, -12               # Desplazamiento en entero
_start:
  movl   %esp, %ebp
```

```
  cmp      $5, ARGC(%ebp)           # Comprobar cantidad de args.
  jne      start_error_parametros   # Error si ARGC != 5

  subl   $12, %esp                  # Espacio para los FD de archivos

  # Abrir archivo de entrada
  movl   $OPEN, %eax                # Codigo de apertura
  movl   A_ENTRADA(%ebp), %ebx      # Enviar nombre de archivo
  movl   $READ_ONLY, %ecx           # Abrir en solo lectura
  movl   $DEFAULT_PER, %edx         # Permisos por defecto
  int      $SYSCALL                 # Llamada a sistema
  cmpl   $0, %eax                   # Comprobar si se abrio correctamente
  jle      start_error_abrir        # Se salta a error si no fue asi
  movl   %eax, A_ENTRADA_FD(%ebp)   # Guardar el codigo de archivo

  # Abrir archivo de salida
  movl   $OPEN, %eax                # Codigo de apertura
  movl   A_SALIDA(%ebp), %ebx       # Enviar nombre de archivo
  movl   $03101, %ecx               # Abrir en escritura, crear, truncar
  movl   $0666, %edx                # Permisos por defecto
  int      $SYSCALL                 # Llamada a sistema
  cmpl   $0, %eax                   # Comprobar si se abrio correctamente
  jle      start_error_abrir        # Se salta a error si no fue asi
  movl   %eax, A_SALIDA_FD(%ebp)    # Guardar el codigo de archivo

  # Revisar el desplazamiento
  pushl  DESPL(%ebp)                # Enviar despl. como parametro
  call   string_int                 # Convertir de texto a numero entero
  addl   $4, %esp                   # Liberar espacio de parametro

  cmp      $0, %eax                 # Si el desplazamiento es menor a cero
  jl       start_error_despl        # es un error
  cmp      $99, %eax                # Si el desplazamiento es mayor a 99
  jg       start_error_despl        # es un error
  movl   %eax, ENT_DESPL(%ebp)      # Se almacena el entero

  # Revisar la bandera
  pushl  BANDERA(%ebp)              # Se envia la bandera como parametro
  call   eod                        # Se comprueba si es -e, -d o error
  addl   $4, %esp                   # Liberar espacio de parametro

  cmp      $1, %eax                 # Si el resultado es 1
  je       start_encriptar          # la bandera es -e y se debe encriptar
  cmp      $2, %eax                 # Si el resultado es 2 la
  je       start_desencriptar       # bandera es -d y se debe desencriptar
  jmp      start_error_bandera      # Si no es 1 ni 2, es error

  # Llamar a la funcion de encriptar
start_encriptar:
  pushl  ENT_DESPL(%ebp)            # Enviar el desplazamiento entero
  pushl  A_SALIDA_FD(%ebp)          # Enviar el FD de salida
  pushl  A_ENTRADA_FD(%ebp)         # Enviar el FD de entrada
  call   encriptar                  # Encriptar contenido de archivo
  addl   $12, %esp                  # Liberar los 3 parametros
  jmp      start_cerrar
```

```
  # Llamar a la funcion de desencriptar
start_desencriptar:
  pushl   ENT_DESPL(%ebp)            # Enviar el desplazamiento entero
  pushl   A_SALIDA_FD(%ebp)          # Enviar el FD de salida
  pushl   A_ENTRADA_FD(%ebp)         # Enviar el FD de entrada
  call    desencriptar               # Liberar los 3 parametros
  addl    $12, %esp

start_cerrar:
  # Cerrar archivo de salida
  movl    $CLOSE, %eax               # Codigo de cerrado
  movl    A_SALIDA_FD(%ebp), %ebx    # Codigo de archivo
  int       $SYSCALL                 # Llamada a sistema

  # Cerrar archivo de entrada
  movl    $CLOSE, %eax               # Codigo de cerrado
  movl    A_ENTRADA_FD(%ebp), %ebx   # Codigo de archivo
  int       $SYSCALL                 # Llamada a sistema

  movl    $0, %ebx                   # No hay errores
  jmp       start_fin                # Saltar al final

start_error_parametros:
  pushl   $ERR_PARAMETROS            # Enviar texto error de parametros
  call    printf                     # Imprimir ese texto
  addl    $4, %esp                   # Liberar espacio de parametro
  call    ayuda                      # Mostrar texto de ayuda
  movl    $1, %ebx                   # Error con la cantidad de parametros
  jmp       start_fin                # Ir al fin del programa

start_error_abrir:
  pushl   $ERR_ARCHIVO               # Enviar texto error de archivo
  call    printf                     # Imprimir ese texto
  addl    $4, %esp                   # Liberar espacio de parametro
  call    ayuda                      # Mostrar texto de ayuda
  movl    $2, %ebx                   # Hubo un error al abrir archivos
  jmp       start_fin                # Ir al fin del programa

start_error_bandera:
  pushl   $ERR_BANDERA               # Enviar texto error de bandera
  call    printf                     # Imprimir ese texto
  addl    $4, %esp                   # Liberar espacio de parametro
  call    ayuda                      # Mostrar texto de ayuda
  movl    $3, %ebx                   # Error en el argumento de bandera
  jmp       start_fin                # Ir al fin del programa

start_error_despl:
  pushl   $ERR_DESPL                 # Enviar texto error de desplazamiento
  call    printf                     # Imprimir ese texto
  addl    $4, %esp                   # Liberar espacio de parametro
  call    ayuda                      # Mostrar texto de ayuda
  movl    $4, %ebx                   # Error en el desplazamiento
  jmp       start_fin                # Ir al fin del programa

start_fin:
  # Finalizar programa
  movl    $1, %eax                   # Codigo de salida
```

```
  int      $SYSCALL                 # Llamada a sistema


  # ____________________________________________________________________________
  # eod ( cadena )
  # Devuelve 1 si la bandera de entrada es igual a -e,
  # 2 si la bandera de entrada es igual a -d
  # y -1 en cualquier otro caso
  .type eod, @function
eod:
  pushl  %ebp
  movl   %esp, %ebp

  xorl   %eax, %eax                # limpia el registro eax
  xorl   %ebx, %ebx                # limpia el registro ebx
  xorl   %edx, %edx                # limpia el registro edx
  xorl   %edi, %edi                # limpia el registro edi

  movl   8(%ebp), %ebx             # carga el parametro de la funcion en ebx
  movb   (%ebx, %edi, 1), %dl      # carga el primer caracter de la cadena

  cmpb   $'-' ,%dl                 # compara si la cadena es correcta
  jl       eod_error
  jg       eod_error

  incl   %edi                      # incrementa edi
  movb   (%ebx, %edi, 1), %dl      # carga el segundo caracter de la cadena

  cmpb   $'e' ,%dl                 # compara si la bandera es -e
  jl       eod_if_d                # salto a la siguiente condicion
  jg       eod_if_d
  movl   $1, %eax                  # carga el valor de retono en eax
  jmp      eod_final               # salto al final de la funcion

eod_if_d:                            # segunda condicion
  cmpb   $'d' ,%dl                 # compara si la bandera de entrada es -d
  jl       eod_error               # la bandera de entrada no es ninguna
  jg       eod_error               # de las esperadas
  movl   $2, %eax                  # carga el valor de retono en eax
  jmp      eod_final               # salto al final de la funcion

  # Si existe un tercer caracter
  incl   %edi                      # incrementa edi
  movb   (%ebx, %edi, 1), %dl      # carga el tercer caracter de la cadena

  cmpb   $0 ,%dl                   # compara si es un valor distinto al final
  jg       eod_error               # de la cadena, si es asi salta a error

eod_error:
  movl   $-1, %eax                 # carga un valor de error en eax

eod_final:
  movl %ebp, %esp
  popl %ebp
  ret
```

```
  # ____________________________________________________________________________
  # printf ( cadena )
  # Imprime en consola un texto
  .type printf, @function
printf:
  pushl  %ebp
  movl   %esp, %ebp

  subl   $4, %esp                   # Obtener espacio para variable local
  movl   $0, -4(%ebp)               # Mover 0 a variable local

  xorl   %eax, %eax                 # limpia el registro eax
  xorl   %ebx, %ebx                 # limpia el registro ebx
  xorl   %edi, %edi                 # limpia el registro edi

  movl   8(%ebp), %ebx              # direccion del texto
                                    # que se quiere imprimir
  movb   (%ebx, %edi, 1), %dl       # primer caracter de la cadena

printf_imp:
  cmpb   $0, %dl                    # fin de la cadena
  je       printf_final             # salto al final de la funcion

  movb   %dl, -4(%ebp)              # caracter a imprimir
  movl   $1, %edx                   # longitud del caracter

  movl   %ebp, %ecx                 # direccion del caracter a imprimir
  subl   $4, %ecx                   # en ebp menos 4

  movl   $1, %ebx                   # identificador de archivo (stdout)
  movl   $4, %eax                   # sys_write (=4)
  int      $0x80                    # llamada a interrupcion de software

  incl   %edi                       # incremento de edi
  movl   8(%ebp), %ebx              # carga la direccion del texto a imprimir
  movb   (%ebx, %edi, 1), %dl       # siguiente caracter de la cadena
  jmp      printf_imp               # salto a inicio del bucle

printf_final:
  movl   %ebp, %esp
  popl   %ebp
  ret


  # ____________________________________________________________________________
  # caracter_int ( caracter )
  # Transforma solo un caracter a numero
 .type caracter_int, @function
caracter_int:
  pushl  %ebp
  movl   %esp, %ebp

  xorl   %eax, %eax                 # limpia el registro eax
  movl   8(%ebp), %eax              # carga el parametro de la funcion en eax

  cmp      $48, %eax                # se compara si el caracter es un número
                                    # menor a cero
```

```
  jl       caracter_int_error        # el caracter no es convertible
  cmpl   $57, %eax                   # compara si el caracter es un número
                                     # mayor que nueve
  jg       caracter_int_error        # el caracter no es convertible

  subl   $48, %eax                   # convierte el caracter numerico a número
  jmp      caracter_int_fin          # salto al final del método

caracter_int_error:
  movl $-1, %eax                     # indica que el caracter no es numerico

caracter_int_fin:
  movl %ebp, %esp
  popl %ebp
  ret

  # ____________________________________________________________________________
  # string_int ( cadena )
  # Transforma una cadena de caracteres a numero
  .type string_int, @function
string_int:
  pushl %ebp
  movl %esp, %ebp

  xorl   %ecx, %ecx                  # limpia el registro ecx
  xorl   %edx, %edx                  # limpia el registro edx
  xorl   %edi, %edi                  # limpia el registro edi
  movl   8(%ebp), %ebx               # carga el parametro de la funcion en ebx
  movb   (%ebx, %edi, 1), %dl        # carga el primer caracter de la cadena

string_int_while:
  cmpb   $0, %dl                     # final de la cadena
  je       string_int_ultimo         # salto a la etiqueta "ultimo"
  pushl  %edx                        # carga el caracter que se decea convertir
  call   caracter_int                # llamada a la funcion convertir caracter
  addl   $4, %esp                    # se libera el espacio asignado en la pila

  cmpl   $-1, %eax                   # si el caracter no fue convertible
  je       string_int_final          # salto al final de la funcion

  imull  $10, %ecx                   # multiplica el resultado almacenado
                                     # para agregar el nuevo digito
  addl   %eax, %ecx                  # se agrega el nuevo digito al total

  incl   %edi                        # incrementa del indice del while
  movb   (%ebx, %edi, 1), %dl        # se actualiza el nuevo caracter
  jmp      string_int_while          # salto al inicio del while

string_int_ultimo:
  movl   %ecx, %eax                  # transferencia de resultado final a eax

string_int_final:
  movl   %ebp, %esp
  popl   %ebp
  ret

  # ____________________________________________________________________________
```

```
  # ayuda ( )
  # Muestra la ayuda en consola
  .type ayuda, @function
ayuda:
  pushl %ebp
  movl %esp, %ebp

  pushl  $TEXTO_AYUDA               # Enviar texto de ayuda como parametro
  call   printf                     # Imprimir texto
  addl   $4, %esp                   # Liberar espacio de parametro

  movl   %ebp, %esp
  popl   %ebp
  ret

  # ____________________________________________________________________________
  # indice ( caracter )
  # Recibe un caracter y obtiene su posicion lineal
  # en la matriz.
  .type indice, @function
indice:
  pushl  %ebp
  movl   %esp, %ebp

  xorl   %eax, %eax                 # Limpiar registro
  movb   ARG1(%ebp), %al            # Mover caracter a registro al

  cmp      $'\t', %al               # Comparar limites, si es menor a '\t'
  jl       indice_fuera             # esta bajo el limite, saltar a fuera
  cmp      $'\r', %al               # Si es mayor que el '\r' el caracter
  jg       indice_2do_bloque        # puede pertenecer al 2do bloque

  subb   $'\t', %al                 # Si esta en el rango, restar el valor de
  jmp      indice_fin               # \t y salir

indice_2do_bloque:
  cmp      $' ', %al                # Si esta bajo el espacio y sobre el '\r'
  jl       indice_fuera             # el caracter no esta en el rango
  cmp      $'~', %al                # Si esta sobre el '~' el caracter esta
  jg       indice_fuera             # sobre el rango

  subb   $' ', %al                  # Si esta en el rango del segundo bloque
  addb   $BLOQUE1, %al              # se resta el valor del espacio y se suma
  jmp      indice_fin               # el total de caracteres en el bloque 1

indice_fuera:
  movl   $-1, %eax                  # Si esta fuera de los bloques, devolver -1

indice_fin:
  movl   %ebp, %esp
  popl   %ebp
  ret

  # ____________________________________________________________________________
  # caracter ( indice )
  # Recibe un indice que representa una posicion linea
  # en la matriz. Devuelve el caracter asignado.
```

```
  .type caracter, @function
caracter:
  pushl  %ebp
  movl   %esp, %ebp

  movl   ARG1(%ebp), %eax           # Pasar indice a eax

  cmpl   $0, %eax                   # Si es menor a cero, esta fuera de rango
  jl       caracter_fuera           # Saltar a error
  cmpl   $4, %eax                   # Si es mayor a 4, puede estar en el 2do
  jg       caracter_2do_bloque      # bloque

  addl   $9, %eax                   # Si esta en el rango, sumar 9 para obtener
  jmp      caracter_fin             # caracter. Ir al final de la funcion

caracter_2do_bloque:
  cmp      $99, %eax                # Si el indice es mayor que 99 el caracter
  jg       caracter_fuera           # esta fuera del rango

  subl   $BLOQUE1, %eax             # Si esta en el 2do bloque, restar el
  addl   $32, %eax                  # tamano del bloque 1 y sumar 32 (espacio)
  jmp      caracter_fin             # Ir al final de la funcion

caracter_fuera:
  movl   $-1, %eax                  # En caso de error, devolver -1

caracter_fin:
  movl   %ebp, %esp
  popl   %ebp
  ret

  # ____________________________________________________________________________
  # caracter_en ( posicion, desplazamiento )
  # Recibe un valor lineal de la tabla y un desplazamiento
  # Devuelve el caracter encriptado
  .equ   VAL_COL, -4
  .equ   VAL_FIL, -8
  .equ   OFF_COL, -12
  .equ   OFF_FIL, -16
  .equ   COL, -20
  .equ   FIL, -24
  .type caracter_en, @function
caracter_en:
  pushl  %ebp
  movl   %esp, %ebp

  subl   $24, %esp                  # Espacio para 6 variables enteras

  movl   $DIM, %ebx                 # Dimension de matriz en ebx

  movl   $0, %edx                   # Limpiar edx
  movl   ARG1(%ebp), %eax           # Parametro de posicion en eax
  divl   %ebx                       # Dividir Posicion entre Dimension
  movl   %eax, VAL_COL(%ebp)        # Cociente en Valor de columna
  movl   %edx, VAL_FIL(%ebp)        # Residuo en Valor de fila

  movl   $0, %edx                   # Limpiar edx
```

```
  movl   ARG2(%ebp), %eax           # Desplazamiento a eax
  divl   %ebx                       # Dividir Desplazamiento entre Dimension
  movl   %eax, OFF_COL(%ebp)        # Cociente a Offset de columna
  movl   %edx, OFF_FIL(%ebp)        # Residuo a Offset de fila

  movl   $0, %edx                   # Limpiar edx
  movl   VAL_FIL(%ebp), %eax        # Valor de fila a eax
  addl   OFF_FIL(%ebp), %eax        # Sumar Offset de fila con Valor de fila
  divl   %ebx                       # Dividir entre Dimension
  movl   %edx, FIL(%ebp)            # Residuo a Fila

  movl   $0, %edx                   # Limpiar edx
  movl   VAL_COL(%ebp), %eax        # Valor de columna a eax
  addl   OFF_COL(%ebp), %eax        # Sumar Offset de columna con Valor columna
  divl   %ebx                       # Dividir entre Dimension
  movl   %edx, COL(%ebp)

  movl   $0, %edx                   # Limpiar edx
  movl   FIL(%ebp), %eax            # Fila a eax
  imull  %ebx                       # Multiplicar por Dimension
  addl   COL(%ebp), %eax            # Sumar Columna

  pushl  %eax                       # Enviar resultado como parametro
  call   caracter                   # Obtener caracter
  addl   $4, %esp                   # Liberar espacio de parametro

  movl   %ebp, %esp
  popl   %ebp
  ret

  # ____________________________________________________________________________
  # caracter_en ( posicion, desplazamiento )
  # Recibe un valor lineal de la tabla y un desplazamiento
  .type caracter_de, @function
caracter_de:
  pushl  %ebp
  movl   %esp, %ebp

  subl   $24, %esp                  # Espacio para 6 variables enteras

  movl   $DIM, %ebx                 # Dimension de matriz en ebx

  movl   $0, %edx                   # Limpiar edx
  movl   ARG1(%ebp), %eax           # Parametro de posicion en eax
  divl   %ebx                       # Dividir entre Dimension
  movl   %eax, VAL_COL(%ebp)        # Cociente en Valor de columna
  movl   %edx, VAL_FIL(%ebp)        # Residuo en Valor de fila

  movl   $0, %edx                   # Limpiar edx
  movl   ARG2(%ebp), %eax           # Parametro de desplazamiento a eax
  divl   %ebx                       # Dividir entre Dimension
  movl   %edx, OFF_COL(%ebp)        # Cociente a Offset de columna
  movl   %eax, OFF_FIL(%ebp)        # Residuo a Offset de fila

  movl   $0, %edx                   # Limpiar edx
  movl   VAL_FIL(%ebp), %eax        # Valor de fila a eax
  subl   OFF_FIL(%ebp), %eax        # Restar Offset de fila del Valor de fila
```

```
  addl   $DIM, %eax                 # Sumarle la Dimension
  divl   %ebx                       # Dividir entre Dimension
  movl   %edx, FIL(%ebp)            # Residuo a Fila

  movl   $0, %edx                   # Limpiar edx
  movl   VAL_COL(%ebp), %eax        # Valor de columna a eax
  subl   OFF_COL(%ebp), %eax        # Restarle el Offset al Valor de columna
  addl   $DIM, %eax                 # Sumarle la Dimension
  divl   %ebx                       # Dividir entre Dimension
  movl   %edx, COL(%ebp)            # Mover residuo a Columna

  movl   $0, %edx                   # Limpiar edx
  movl   FIL(%ebp), %eax            # Fila a eax
  imull  %ebx                       # Multiplicar por Dimension
  addl   COL(%ebp), %eax            # Sumar Columna

  pushl  %eax                       # Enviar resultado como parametro
  call   caracter                   # Obtener caracter
  addl   $4, %esp                   # Liberar espacio de parametro

  movl   %ebp, %esp
  popl   %ebp
  ret

  # ____________________________________________________________________________
  # Encriptar ( archivo_entrada, archivo_salida, desplazamiento )
  # Recibe el FD de los archivos de entrada y salida y el
  # desplazamiento

# Argumentos
  .equ ENTRADA_FD, 8                  # Direccion de archivo de entrada
  .equ SALIDA_FD, 12                  # Direccion de archivo de salida
  .equ DESPLAZAMIENTO, 16             # Desplazamiento de encripcion
# Variables Internas
  .equ TOTAL_CARACTERES, -4           # Caracteres leidos al buffer

  .type encriptar, @function
encriptar:
  pushl  %ebp
  movl   %esp, %ebp

  subl   $4, %esp                   # Almacenar espacio para variable local

encriptar_buffer_inicio:
  # Leer un trozo de texto
  movl   $READ, %eax                # Codigo de lectura
  movl   ENTRADA_FD(%ebp), %ebx     # Enviar el descriptor de entrada
  movl   $BUFFER, %ecx              # Buffer a donde se leera
  movl   $TBUFFER, %edx             # Tamano del buffer para leer
  int      $SYSCALL                 # Llamada a sistema
  cmp      $0, %eax                 # Comparar si se leyeron caracteres
  jle      encriptar_fin            # Si no, se sale

  pushl  %eax                       # Guardar el total de caracteres leidos

  xorl   %edi, %edi                 # Limpiar el contador numerico
  movl   %eax, TOTAL_CARACTERES(%ebp) # Guardar el total de caracteres en
```

```
                                    # variable local
encriptar_bloque:
  movl   $BUFFER, %ebx              # Guardar la direccion del buffer
  xorl   %eax, %eax                 # Limpiar eax
  movb   (%ebx, %edi, 1), %al       # Mover caracter a %al

  pushl  %eax                       # Enviar el caracter leido a indice
  call   indice                     # Obtener el indice
  addl   $4, %esp                   # Limpiar el espacio del argumento

  pushl  DESPLAZAMIENTO(%ebp)       # Enviar el desplazamiento
  pushl  %eax                       # Enviar posicion (indice)
  call   caracter_en                # Encriptar caracter
  addl   $8, %esp                   # Limpiar argumentos

  movl   $BUFFER, %ebx              # Guardar direccion del buffer
  movb   %al, (%ebx, %edi, 1)       # Mover caracter encriptado

  incl   %edi
  cmpl   %edi, TOTAL_CARACTERES(%ebp) # Comparar el indice con total
  jl       encriptar_bloque_fin     # Se termino de encriptar bloque
  jmp      encriptar_bloque         # Si no, seguir encriptando

encriptar_bloque_fin:
  popl   %edx                       # Total de caracteres leidos
  movl   $WRITE, %eax               # Codigo de escritura
  movl   SALIDA_FD(%ebp), %ebx      # Enviar el archivo de salida
  movl   $BUFFER, %ecx              # Enviar direccion del buffer
  int      $SYSCALL                 # Llamada al sistema
  jmp      encriptar_buffer_inicio  # Seguir encriptando

encriptar_fin:
  movl   %ebp, %esp
  popl   %ebp
  ret


  # ____________________________________________________________________________
  # Desencriptar (archivo_entrada, archivo_salida, desplazamiento)
  # Recibe el FD de los archivos de entrada y salida y el
  # desplazamiento

# Argumentos
  .equ ENTRADA_FD, 8                  # Posicion del archivo de entrada
  .equ SALIDA_FD, 12                  # Posicion del archivo de salida
  .equ DESPLAZAMIENTO, 16             # Desplazamiento de desencripcion
# Variables Internas
  .equ TOTAL_CARACTERES, -4           # Caracteres leidos

  .type desencriptar, @function
desencriptar:
  pushl  %ebp
  movl   %esp, %ebp

  subl   $4, %esp   # Almacenar espacio para variable local

desencriptar_buffer_inicio:
```

```
  # Leer un trozo de texto
  movl   $READ, %eax                # Codigo de lectura
  movl   ENTRADA_FD(%ebp), %ebx     # Enviar el descriptor de entrada
  movl   $BUFFER, %ecx              # Buffer al que se leera
  movl   $TBUFFER, %edx             # Tamano del buffer para leer
  int      $SYSCALL                 # Llamada a sistema
  cmp      $0, %eax                 # Comparar si se leyeron caracteres
  jle      desencriptar_fin         # Si no, se sale

  pushl  %eax                       # Guardar el total de caracteres leidos

  xorl   %edi, %edi                 # Limpiar el contador numerico
  movl   %eax, TOTAL_CARACTERES(%ebp) # Guardar el total de caracteres en
                                    # variable local
desencriptar_bloque:
  movl   $BUFFER, %ebx              # Guardar la direccion del buffer
  xorl   %eax, %eax                 # Limpiar eax
  movb   (%ebx, %edi, 1), %al       # Mover caracter a %al

  pushl  %eax                       # Enviar el caracter leido a indice
  call   indice                     # Obtener el indice
  addl   $4, %esp                   # Limpiar el espacio del argumento

  pushl  DESPLAZAMIENTO(%ebp)       # Enviar el desplazamiento
  pushl  %eax                       # Enviar posicion (indice)
  call   caracter_de                # Encriptar caracter
  addl   $8, %esp                   # Limpiar argumentos

  movl   $BUFFER, %ebx              # Guardar direccion del buffer
  movb   %al, (%ebx, %edi, 1)       # Mover caracter encriptado

  incl   %edi
  cmpl   %edi, TOTAL_CARACTERES(%ebp) # Comparar el indice con total
  jl       desencriptar_bloque_fin  # Se llego al final del buffer
  jmp      desencriptar_bloque      # Si no, sigue desencriptando

desencriptar_bloque_fin:
  popl   %edx                       # Total de caracteres leidos
  movl   $WRITE, %eax               # Codigo de escritura
  movl   SALIDA_FD(%ebp), %ebx      # Seleccionar el archivo de salida
  movl   $BUFFER, %ecx              # Se envia el buffer desencriptado
  int      $SYSCALL                 # Llamada al sistema
  jmp      desencriptar_buffer_inicio # Seguir leyendo desde arriba

desencriptar_fin:
  movl   %ebp, %esp
  popl   %ebp
  ret

  # FIN DEL PROGRAMA #
```