INSTITUTO TECNOLÓGICO DE MORELIA

LENGUAJES DE INTERFAZ

REPORTE. - PRÁCTICA DE CADENAS

PROFESOR: JORGE LUIS DÍAZ HUERTA

ALUMNOS:

(17120173) SEBASTIAN HERNÁNDEZ PEDRAZA

(17120160) CÉSAR EDUARDO ESPINOSA MIRANDA

GRUPO: A

FECHA DE ENTREGA: 12 DE DICIEMBRE DE 2020

Introducción

A continuación, se presentará el siguiente código, el cual tiene funcionalidades de manipulación de cadenas, como lo son: Espejo (pone la cadena al revés), convierte todos los caracteres a minúsculas, convierte todos los caracteres a mayúsculas y por último, convierte los caracteres en minúscula a mayúscula y las mayúsculas a minúsculas. El código se irá explicando a con una imágen de referencia.

PRÁCTICA - MANIPULACIÓN DE CADENAS

```
org 100h
.DATA

str0 db 10,13, "BIENVENIDO AL MANIPULADOR DE CADENAS $"
str1 db 0Dh, 0Ah, 0Dh, 0Ah, "Introduce tu cadena: $"
msg4 db 0Dh, 0Ah, 0Dh, 0Ah, "Funcion Espejo [1] $"
msg1 db 0Dh, 0Ah, 0Dh, 0Ah, "Convertir a Minusculas [2] $"
msg2 db 0Dh, 0Ah, 0Dh, 0Ah, "Convertir a Mayusculas [3] $"
msg3 db 0Dh, 0Ah, 0Dh, 0Ah, "Inversion de caracteres [4] $"
msg6 db 0Dh, 0Ah, 0Dh, 0Ah, "Salir [5] $"
msg5 db 0Dh, 0Ah, 0Dh, 0Ah, "Introduce la opcion: $"

;str2 db 10,13, 'Output: $'
numCont db 0 ;guarda el num de cont
```

Primeramente, comenzamos declarando los mensajes que vamos a mostrar.

La variable de numCont, nos servirá para guardar la opción elegida.

```
.CODE
mov ax, @data ;se carga el data al acumulador
mov ds, ax

mov dx, offset str0 ;opciones
mov ah, 9
int 21h

;org 100h
mov dx, offset str1 ;cadena de entrada
mov ah, 9
int 21h

mov dx, offset buffer
mov ah, 0ah
int 21h
jmp print
buffer db 20,?, 20 dup(' ')

print:
xor bx, bx
mov bl, buffer[1]
mov buffer[bx+2], '$'
```

Después mandamos llamar las variables de nuestros mensajes, en donde primeramente se desplegará el mensaje de bienvenida, después, se imprimirá el mensaje donde ya nos pedirá la cadena como tal, y será guardada en buffer. Después, le decimos que el tamaño de nuestro buffer será de 20.

Después en "print", se le pone el $ al final de nuestra cadena (esto para que la cadena tenga un límite de términos). Si no ponemos el bx+2, nos imprimirá dos caracteres menos de nuestra cadena.

```
mensajes:
mov dx, offset msg4 ;opciones
mov ah, 9
int 21h

mov dx, offset msg1 ;opciones
mov ah, 9
int 21h

mov dx, offset msg2 ;opciones
mov ah, 9
int 21h

mov dx, offset msg3 ;opciones
mov ah, 9
int 21h

mov dx, offset msg6 ;opciones
mov ah, 9
int 21h

mov dx, offset msg5 ;opciones
mov ah, 9
int 21h
```

En esta parte, imprimimos en pantalla todos los mensajes de las opciones que podemos elegir.

```
mov ah, 01
int 21h
mov numCont, al
```

Aquí, primeramente con la interrupción 21H, AH=1, estamos leyendo la opción que el usuario ingrese por teclado. Después lo que quede en "al" lo mandamos a nuestra variable numCont.

```
mov dx,13
mov ah,2
int 21h
mov dx,10
mov ah,2
int 21h
```

Esto es solamente un salto de línea.

```
;Opciones
cmp numCont, 31h ;31h es 1 en hexadecimal
JE espejo

cmp numCont, 32h ;31h es 1 en hexadecimal
JE aminus

cmp numCont, 33h
JE amayus

cmp numCont, 34h
JE invierte

cmp numCont, 35h
JE salir
```

Después, se hará la comparación de la opción introducida por el usuario previamente, si escogió la opción 1 (31h en Hexadecimal) saltará a la función “espejo”, y así consecutivamente con cada función que fue creada para cada opción.

```
espejo:
    ; carga donde esta la cadena
    mov si, offset buffer

    ; contador de los caracteres de la cadena
    mov cx, 0h

    pusheador:
    ; Para comparar si es el ulitmo caracter de la cadena
    mov ax, [si]
    cmp al, '$'
    je cargaAgain

    ; Pushea en la pila en caso de que aun no acabe la pila
    push [si]

    ; Se incrementa el contador y el SI para incrementar el indice
    inc si
    inc cx

    jmp pusheador

    cargaAgain:
    ; se carga de nuevo la posicion de la cadena inicial
    mov si, offset buffer

    popeador:
    ;cuando el contador llegue a 0 imprime la cadena
    cmp cx,0h
    je imprimeEspejo

    ; hace pop en la punta de la fila
    pop dx

    ; hacemos dh 0
    mov dh, 0

    ; vamos poniendo caracter por caracter
    mov [si], dx

    ; se incrementa el indice y le restamos al contador
    inc si
    dec cx

    jmp popeador
```

Para el método “espejo”, es en donde cumpliremos la función de invertir la cadena ingresada por el usuario, primeramente vamos a mover a SI el valor de “buffer” para después utilizarlo, después limpiaremos el registro cx para tenerlo en 0. Primeramente, antes de revisar la cadena, se revisa que no sea el fin de esta, utilizando así el “cmp” para verificar que el caracter leido en ese momento no sea un “$” (fin de cadena), y en caso de

que no sea asi, continuara con la ejecución de este método, llegando a la parte de push[si], que es donde estamos ya ingresando el valor de SI en ese momento a la pila. Una vez que nuestro valor se encuentra en la pila, se incrementa el contador “cx” y el índice de la pila “SI”, que nos servirá para ir recorriendo la pila e ir pusheando los valores que estén en la posición que tenga SI. Al final, se vuelve a repetir la función anteriormente explicada hasta encontrar el ya mencionado “$”

Ahora, en la función cargaAgain, se reinicia el valor de sí, dándole de nuevo el valor del buffer.

Después en el método popeador, primero se revisará que el valor del contador no sea 0, después se ejecutará la función “pop” para ir sacando los valores de la pila, limpiamos dh, luego, vamos poniendo caracter por caracter, incrementamos nuestro índice de pila para ir recorriendo y decrementamos cx hasta llegar al último caracter de nuestra cadena.

```
imprimeEspejo:
; para que no siga imprimiendo cosas raras
mov [si],'$'

; imprimimos
lea dx,buffer
mov ah, 09H
int 21H

jmp mensajes
```

En esta parte de “imprimeEspejo”, le agregamos un signo de pesos “$”, para evitar que siga imprimiendo caracteres que no queremos. Después, mediante el “lea dx, buffer”, imprimimos nuestra cadena. Y por último, hace un salto a “mensajes”, para mostrar nuevamente las opciones de manipulación de la cadena.

```
aminus:
    mov bx,offset buffer

    mov cx,20

    l1:
    cmp [bx],20h
    je espacio

    ;Por si ya es minuscula no la cambie
    cmp [bx], 60H
    ja comp2
    jna sumale
    comp2:
    cmp [bx], 7BH
    jb espacio

    sumale:
    add [bx],32

    espacio:

    mov dx, [bx]
    ; cmp dx,41
    ; jb incre
    mov ah,02  ;imprime
    int 21h
    ;incre:
    inc bx
    loop l1
    jmp mensajes
```

Aquí llegamos a la opción de convertir nuestra cadena en minúsculas. Primero cargamos a bx nuestra cadena, movemos 20 a cx, que es el tamaño de nuestro buffer. En “l1”

comenzamos con una comparación, para cuando haya espacios entre palabras, y en caso de haberlo salta a “espacio”, el cual se explicará más adelante. Después compara lo que tenemos en [bx] con 60H, para ver si es una letra minúscula y no la cambie. En caso de que sea mayor de 60H, salta a comp2 con un jump above, y en este compara que el valor sea menor de 7BH, que es el último dato en minúscula en valor ASCII, con esto tenemos el rango de (60H a 7BH, y ahí están las letras de la “a” a la “z” en minúsculas). En caso de que el valor del caracter sea menor a 60H (mayúscula), mediante un jump not above salta a “sumale”, el cual le sumará 32 al valor del caracter, convirtiéndolo a minúscula. En “espacio”, vamos a ir imprimiendo caracter por caracter, incrementando bx, para ir avanzando en la posición de nuestra pila, y mediante un loop, volveremos a ejecutar todas las instrucciones y cuando terminemos saltamos a mostrar los mensajes de las opciones, de nuevo.

```
amayus:
 mov bx,offset buffer

    mov cx,20

    l12:
    cmp [bx],20h
    je espacio2

    ;Por si ya es mayuscula no la cambie
    cmp [bx], 40H
    ja comp22
    jna sumale2
    comp22:
    cmp [bx], 5BH
    jb espacio2

    sumale2:
    sub [bx],32

    espacio2:
    mov dx, [bx]

    mov ah,02  ;imprime
    int 21h

    inc bx
    loop l12
    jmp mensajes
```

Aquí llegamos a la opción de convertir nuestra cadena en minúsculas. Primero cargamos a bx nuestra cadena, movemos 20 a cx, que es el tamaño de nuestro buffer. En “l12” comenzamos con una comparación, para cuando haya espacios entre palabras, y en caso de haberlo salta a “espacio2”, el cual se explicará más adelante. Después compara lo que tenemos en [bx] con 40H, para ver si es una letra mayúscula y no la cambie. En caso de que sea mayor de 60H, salta a comp22 con un jump above, y en este compara que el valor sea menor de 5BH, que es el último dato en mayúscula en valor ASCII, con esto tenemos el rango de (40H a 5BH, y ahí están las letras de la “A” a la “Z” en mayúsculas). En caso de que el valor del caracter sea menor a 40H (minúscula), mediante un jump not above salta a “sumale2”, el cual le restará 32 al valor del caracter, convirtiéndolo a mayúscula. En “espacio”, vamos a ir imprimiendo caracter por caracter, incrementando bx, para ir

avanzando en la posición de nuestra pila, y mediante un loop, volveremos a ejecutar todas las instrucciones y cuando terminemos saltamos a mostrar los mensajes de las opciones, de nuevo.

```
invierte:
    mov bx,offset buffer

    mov cx,20

    l3:
    cmp [bx],20h
    je espacio3

    ;mayus-minus
    cmp [bx],40H
    ja otroCom1

    otroCom1:
    cmp [bx], 5BH
    jb sumale3
    jnb otroCom2

    sumale3:
    add [bx],32
    jmp espacio3

    ;minus-mayus
    otroCom2:
    cmp [bx],60H
    ja otroCom3

    otroCom3:
    cmp [bx], 7AH
    jb resta4
    jnb espacio3

    resta4:
    sub [bx],32

    espacio3:
    mov dx, [bx]

    mov ah,02  ;imprime
    int 21h

    inc bx
    loop l3
    jmp mensajes
```

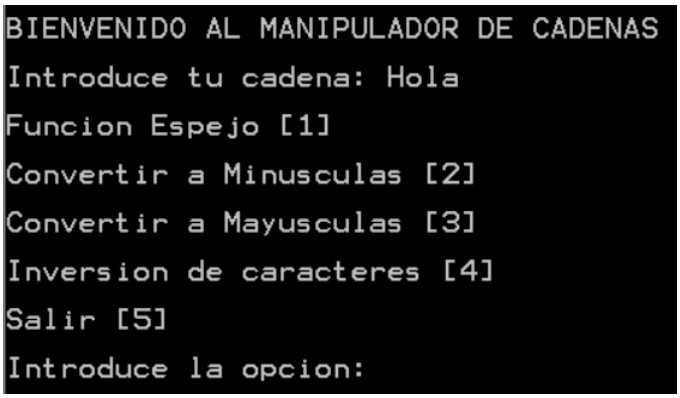
Ahora llegamos a “invierte”, donde en caso de ser mayúscula convertimos a minúscula y viceversa. Movemos a bx nuestra cadena volvemos a poner en el contador el tamaño máximo de nuestro buffer, en “l3” comparamos que no sea un espacio y en caso de serlo salta a “espacio3”, luego comparamos el valor de nuestro caracter, donde si es mayor de 40H salta a “otroCom1” y ahora compara si es menor de 5BH, en caso de ser menor hace un jump below a “sumale3”, donde se le sumará 32 para convertir a minúscula el caracter. En caso de que el valor no sea menor, hace un jump not below a “otroCom2”, donde ahora comparará con 60H y en caso de ser mayor hace un jump above a “otroCom3”. Aquí, compara ahora con 7AH y si es menor, entonces lo manda a “resta4”, donde se le restará 32 para convertirlo de minúscula a mayúscula. En caso de que no sea menor hace salto a “espacio3” y posteriormente imprime caracter por caracter, incrementando bx y ejecutándose loop l3 hasta terminar y hace salto a mensaje de las opciones.

Conclusiones

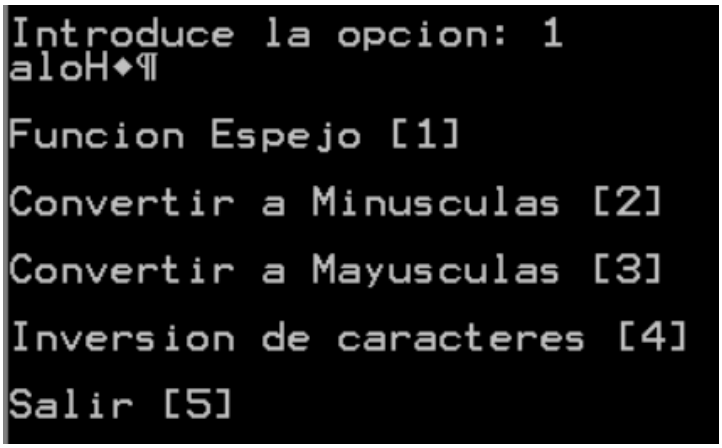
La práctica fue una de las que más hemos tardado haciendo, ya que la parte que más se nos complicó fue la de la función de espejo, debido a que no sabíamos muy bien como hacerla, pero después de estar un rato probando nos funcionó. A su vez, las demás funciones fueron un tanto más sencillas y no hubo más complicación. El único problema que tuvimos y que desafortunadamente no pudimos solucionar, fue que nos imprimía caracteres de más en nuestra cadena de resultado. A pesar de estar checando parte por parte no logramos corregirlo. Sin embargo, el funcionamiento de todas las funciones es el correcto.

La práctica en general fue interesante, ya que aprendimos a manipular las cadenas con ensamblador y no de la forma que usualmente lo hacemos en lenguajes de programación como Java, Python, etc.

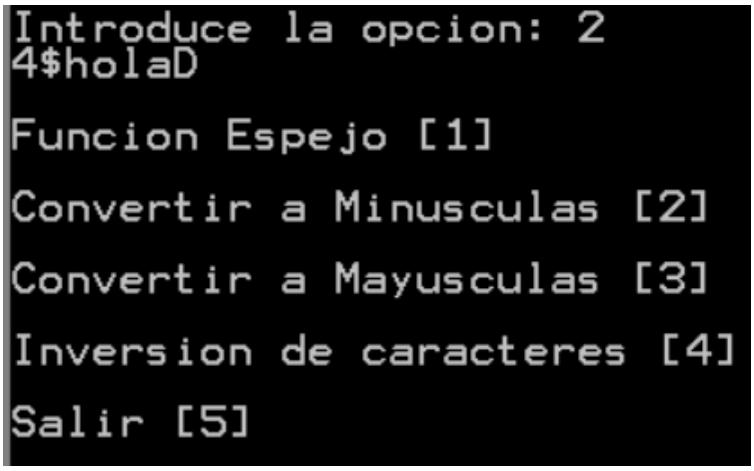
Ingreso de cadena y menú



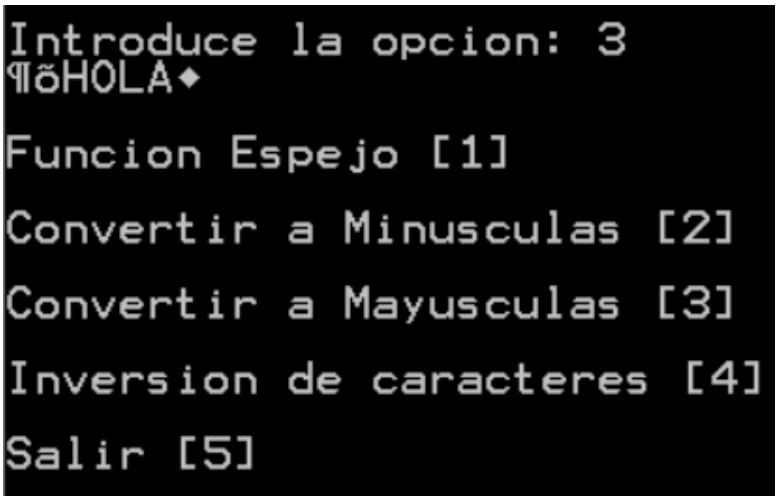
Ejecución de función espejo



Conversión a minúsculas



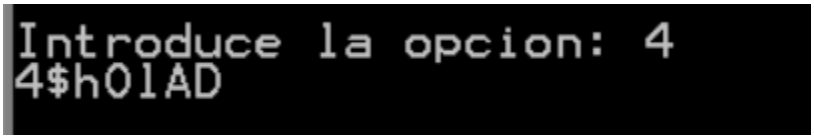
Conversión a mayúsculas



Inversión de carcateres

Cadena ingresada

Resultado



Código del programa

```
org 100h
.DATA

str0 db 10,13, "BIENVENIDO AL MANIPULADOR DE CADENAS $"
str1 db 0Dh, 0Ah, 0Dh, 0Ah, "Introduce tu cadena: $"
msg4 db 0Dh, 0Ah, 0Dh, 0Ah, "Funcion Espejo [1] $"
msg1 db 0Dh, 0Ah, 0Dh, 0Ah, "Convertir a Minusculas [2] $"
msg2 db 0Dh, 0Ah, 0Dh, 0Ah, "Convertir a Mayusculas [3] $"
msg3 db 0Dh, 0Ah, 0Dh, 0Ah, "Inversion de caracteres [4] $"
msg6 db 0Dh, 0Ah, 0Dh, 0Ah, "Salir [5] $"
msg5 db 0Dh, 0Ah, 0Dh, 0Ah, "Introduce la opcion: $"


;str2 db 10,13, 'Output: $'

numCont db 0 ;guarda el num de cont


; -------------------------------------------------
.CODE
mov ax, @data ;se carga el data al acumulador
mov ds, ax

mov dx, offset str0 ;opciones
mov ah, 9
int 21h

;org 100h
```

```
mov dx, offset str1 ;cadena de entrada
mov ah, 9
int 21h

mov dx, offset buffer
mov ah, 0ah
int 21h
jmp print
buffer db 20,?, 20 dup(' ')

print:
xor bx, bx
mov bl, buffer[1]
mov buffer[bx+2], '$'

mensajes:
mov dx, offset msg4 ;opciones
mov ah, 9
int 21h

mov dx, offset msg1 ;opciones
mov ah, 9
int 21h

mov dx, offset msg2 ;opciones
mov ah, 9
int 21h

mov dx, offset msg3 ;opciones
```

```
mov ah, 9
int 21h

mov dx, offset msg6 ;opciones
mov ah, 9
int 21h

mov dx, offset msg5 ;opciones
mov ah, 9
int 21h

mov ah, 01
int 21h
mov numCont, al

mov dx,13
mov ah,2
int 21h
mov dx,10          ;  Todo es un salto de linea
mov ah,2
int 21h

;Opciones
cmp numCont, 31h ;31h es 1 en hexadecimal
JE espejo

cmp numCont, 32h ;31h es 1 en hexadecimal
JE aminus
```

```
cmp numCont, 33h
JE amayus

cmp numCont, 34h
JE invierte

cmp numCont, 35h
JE salir



;-----------------------------------------------------
espejo:
        ; carga donde esta la cadena
        mov si, offset buffer

        ; contador de los caracteres de la cadena
        mov cx, 0h

        pusheador:
        ; Para comparar si es el ulitmo caracter de la cadena
        mov ax, [si]
        cmp al, '$'
        je cargaAgain

        ; Pushea en la pila en caso de que aun no acabe la pila
        push [si]

        ; Se incrementa el contador y el SI para incrementar el indice
        inc si
```

```
inc cx

jmp pusheador

cargaAgain:
; se carga de nuevo la posicion de la cadena inicial
mov si, offset buffer

popeador:
;cuando el contador llegue a 0 imprime la cadena
cmp cx,0h
je imprimeEspejo

; hace pop en la punta de la fila
pop dx

; hacemos dh 0
mov dh, 0

; vamos poniendo caracter por caracter
mov [si], dx

; se incrementa el indice y le restamos al contador
inc si
dec cx

jmp popeador
```

```
imprimeEspejo:
; para que no siga imprimiendo cosas raras
mov [si],'$'

; imprimimos
lea dx,buffer
mov ah, 09H
int 21H

jmp mensajes



aminus:
mov bx,offset buffer

mov cx,20

l1:
cmp [bx],20h
je espacio

;Por si ya es minuscula no la cambie
cmp [bx], 60H
ja comp2
jna sumale
comp2:
cmp [bx], 7BH
jb espacio
```

```
        sumale:
        add [bx],32

        espacio:

        mov dx, [bx]
        ; cmp dx,41
        ; jb incre
        mov ah,02  ;imprime
        int 21h
        ;incre:
        inc bx
        loop l1
        jmp mensajes


amayus:
 mov bx,offset buffer

        mov cx,20

        l12:
        cmp [bx],20h
        je espacio2

        ;Por si ya es mayuscula no la cambie
        cmp [bx], 40H
        ja comp22
        jna sumale2
```

```
        comp22:
        cmp [bx], 5BH
        jb espacio2

        sumale2:
        sub [bx],32

        espacio2:
        mov dx, [bx]

        mov ah,02  ;imprime
        int 21h

        inc bx
        loop l12
        jmp mensajes


invierte:
        mov bx,offset buffer

        mov cx,20

        l3:
        cmp [bx],20h
        je espacio3

        ;mayus-minus
        cmp [bx],40H
```

```
ja otroCom1

otroCom1:
cmp [bx], 5BH
jb sumale3
jnb otroCom2

sumale3:
add [bx],32
jmp espacio3

;minus-mayus
otroCom2:
cmp [bx],60H
ja otroCom3

otroCom3:
cmp [bx], 7AH
jb resta4
jnb espacio3

resta4:
sub [bx],32

espacio3:
mov dx, [bx]

mov ah,02  ;imprime
int 21h
```

```
        inc bx
        loop l3
        jmp mensajes


salir:
.exit
```