ENVIAR POR EMAIL

Apellido y Nombre: Chiola Valentino Hojas:

1	2						TOTAL
4/10	6/10						10/10
3+	2-						6 (Seis)

Totalizador Assembler 14/JUL/2025

DESARROLLAR LOS SIGUIENTES EJERCICIOS EN ASSEMBLER DE LA MV IMPLEMENTADA EN EL CURSO, INCLUYENDO UNA **ESPECIFICACIÓN DE INVOCACIÓN PARA CADA SUBRUTINA**. **RESOLVER CADA EJERCICIO EN UNA HOJA DIFERENTE**.

Ejercicio 1: Traducir el siguiente código en C a ASM de la máquina virtual.

```c
typedef struct _btn{
    short int value;
    struct _btn* left;
    struct _btn* right;
} node;
```

btn

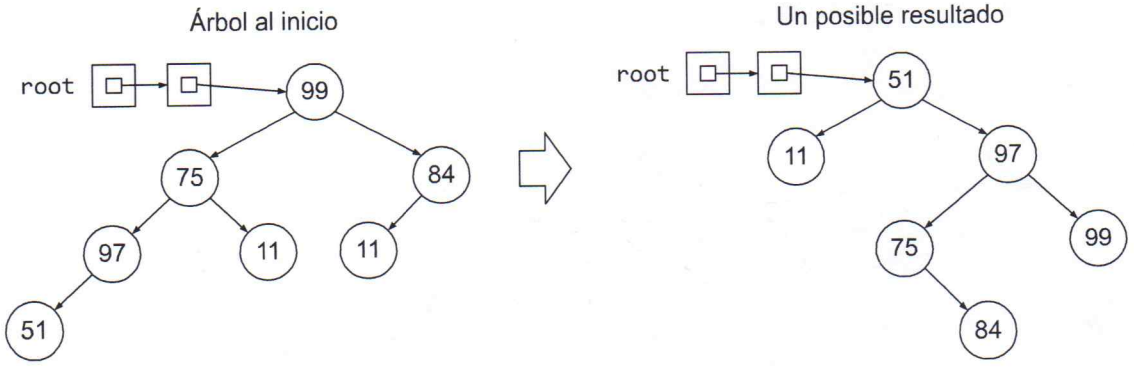
```c
btn** find_pos (btn** root, short int value){
    register btn** aux = root;
    while (*aux != NULL && (*aux)->value != value){
        if ((*aux)->value < node->value) {
            aux = &(*aux)->right;
        } else {
            aux = &(*aux)->left;
        }
    }
    return aux;
}
```

Ejercicio 2: Utilizando la subrutina del ejercicio 1, desarrollar la subrutina "**BT_TO_SEARCH**", y todas las necesarias, para que dado un árbol binario (**NO ordenado**), **lo convierta en un árbol binario de búsqueda.** Si hay nodos repetidos debe eliminarlos. Deben utilizarse los mismos nodos del árbol original, sin crear nodos nuevos.

La subrutina principal debe corresponderse con el siguiente encabezado en C:

```c
void bt_to_search (btn** root)
```

Ejemplo:



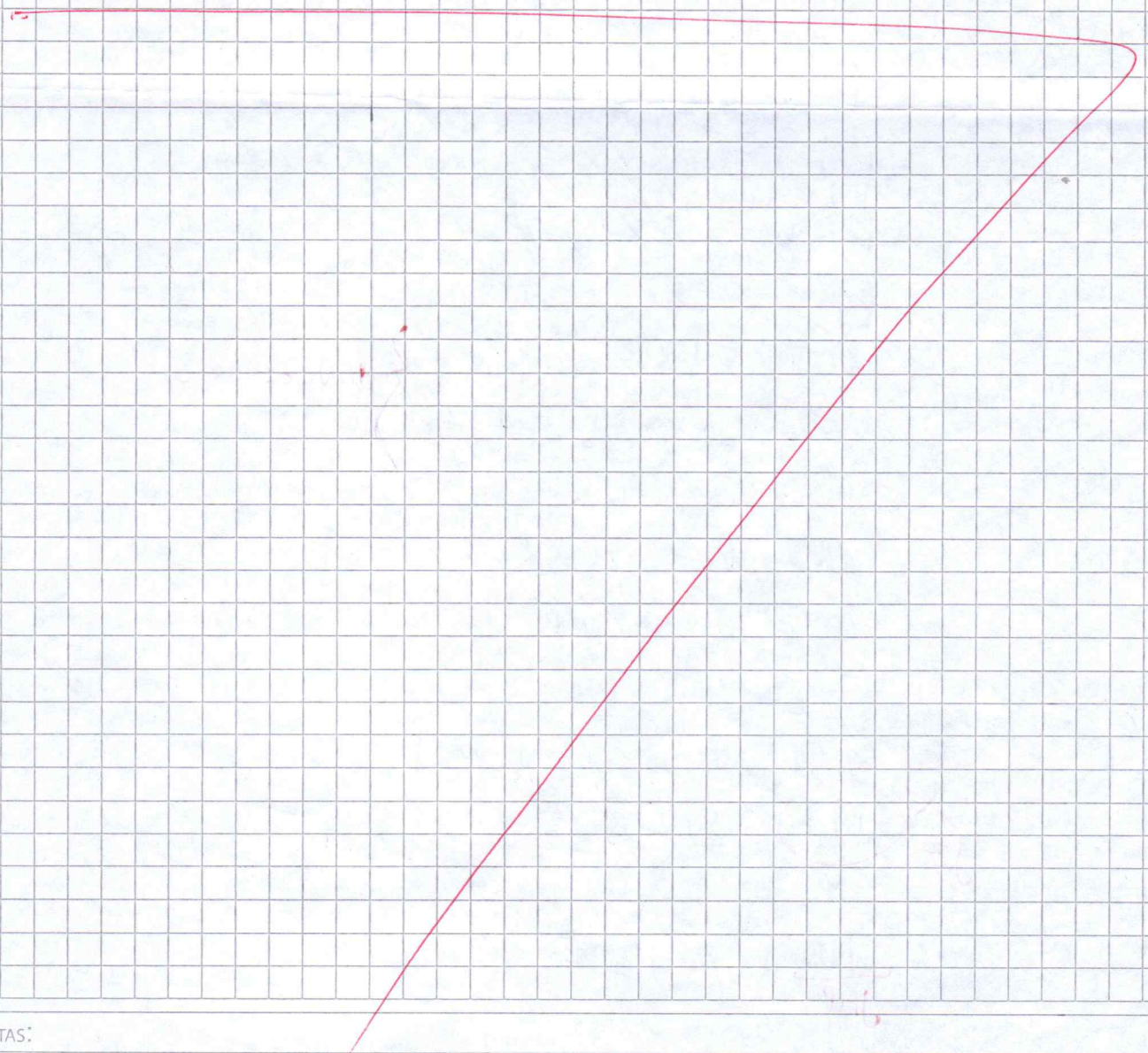
NOTA: Asumir que existe la subrutina "**FREE**" para liberar memoria dinámica.

NOTA: Se exige un mínimo del 40% de desarrollo correcto de cada ejercicio para comenzar a tener puntaje del mismo.
Realizar cada ejercicio en forma completa, precisa y con letra legible.

```
null  equ  -1

value equ 0
left  equ 2
right equ 6

; Push <short int value> +12
; Push <btn**root>  +8
; Call Find_pos
; add sp, 8
; eax-> btn**

Find_pos: Push bp
          mov bp, sp
          Push edx ; **root
          Push ebx ; *aux
          Push cx  ; value
          mov  edx, [bp+8]   } INNECESARIO
          mov  cx , w[bp+14] }
          mov eax, edx
while:    mov ebx, [eax]
          cmp ebx, null
          jz  sigue
          cmp w[ebx+value], cx
          JNZ  sigue        (JZ tachado)
          JN  Por_der
          JNN Por_Izq       (JMP)
```

NOTAS:

```
por_der:    mov eax, ebx
            add eax, right
            jmp while
por_izq:    mov eax, ebx
            add eax, left
            jmp while
sigue:      pop cx
            pop ebx
            pop edx
            mov sp, bp
            pop bp
            Ret
```



NOTAS:

Valentino Chiola

```
value equ 0
left  equ 2
right equ 6
```

```
;Push <btn** root>  +8
;Call bt_to_search
;add  SP, 4
;void
```

```
; Push <bt** orig>  +12
; Push <btn** target> +8
; Call _bt_to_search
; add SP, 8          ; void
```

```
bt_to_search: Push bP
              mov bP, SP
              Push edX   ; ** target
              Sub SP, 4
              mov edx, bp
              sub edx, 4
              mov [edx], null
              mov ebx, [bP+8]
              push ebx
              push edX
              Call _bt_to_search
              add SP, 8
              mov [ebx], [edx]
              add SP, 4
              pop ebx
              pop edx
              mov SP, bP
              pop bP
              Ret
```

despues del mov bp,sp

Push ebx;

NOTAS:

```
_bt_to_search: Push bP
               mov bP,sP
               Push ebx ; original
               Push edx ; target
               Push eax
               push efx ; *aux
               mov ebx, [bP+12]
               mov edx ; [bP+8]
               mov efx, [ebx]
               cmp efx, null
               jz Fin
               push w[efx+value] ; short int
               push edx ; * target
               call find_pos
               add sp, 8
               ; Primero me desplazo a los hijos
               ; luego elimino e inserto
               mov ebx, efx
               add ebx, left
               push ebx
               push edx
               call _bt_tosearch
               add sp, 8
               sub ebx, left
               add ebx, right
               Push ebx
               Push edx
               Call _bt_to_search
               add sp, 8
```

"desenlazar del Padre" 2

mov [ebx] NULL (otro modo de hacer 2)

1

Se debe usar después.

Las invocaciones recursivas por izq y der también buscan ubicación en el TARGET.

Si la misma no se utiliza, varios "instancios" encuentran lo mismo

(*) Invocaciones a la subrut find_pos.

Aquí llamar a FIND. POS 1

NOTAS:

```
        cmp [eax], NULL
        jz  inserta
        jNZ elimm

elimm:  push [eax]
        call free
        add  sp, 4
        jmp  Fin

inserta: push efx
        push edx
        call ins_ord
        add  sp, 8

Fin:    pop  efx
        pop  eax
        pop  edx
        pop  ebx
        mov  sp, bp
        pop  bp
        ret

;Push <**lista> + 12
;push <**target> + 8
;Call  ins_ord: Push bp
                mov  bp, sp
                Push edx  ; ** target
                Push ebx  ; * ebx
                Push efx  ; * aux
```

No requiere otro Funcion. Find pos devuelve el LUGAR DONDE debe insertar

falto ~~poner el padre~~ "desenlazar del padre"

(2) | mov eax [bp+12]
mov [eax] NULL

IMPORTANTE!!!

NOTAS:

```
        mov  edx, [bp + 8]
        mov  ebx, [bp+12]
        mov  efx, [edx]
        cmp  efx, null
        jz   inserta
        cmp  w[ebx + value], w[efx + value]
        jg   por_der
        jmp  por_izq
por_der: push [efx + right]
        push edx
        call ins_ord
        add  sp, 8
        jmp  fin
por_izq: push [efx + left]
        push edx
        call ins_ord
        add  sp, 8
fin:    pop  efx
        pop  ebx
        pop  edx
        mov  sp, bp
        pop  bp
        ret
```

[bp+12] → Se debe mover en el target

[bp+12] → Se debe mover en el TARGET

- Recorre en post orden - por lo tanto find_pos lo debe usar despues
- No es necesaria la funcion ins_ord, pero ademas tiene errores graves

NOTAS: - No "desvincula" cada nodo del arbol original