UNIVERSIDAD DE GUADALAJARA
CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍA
DIVISION DE ELECTRONICA Y COMPUTACION
DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

Practica 6 | Análisis Léxico

SEMINARIO DE SOLUCIÓN DE PROBLEMAS DE TRADUCTORES DE LENGUAJE I (I7026)
Sección: D01

Profesor: Valentín Martínez López
Bryan Eduardo Zarate Miramontes
Código: 217444026
17/05/2024

Objetivo: El alumno deberá realizar un programa que obtenga una cadena desde el teclado que representará una ecuación aritmética cuyos datos serán de 16 bits con signo (deben considerarse mínimo 4 datos). El programa debe resolver la ecuación e indicar su resultado también en 16 bits con signo.

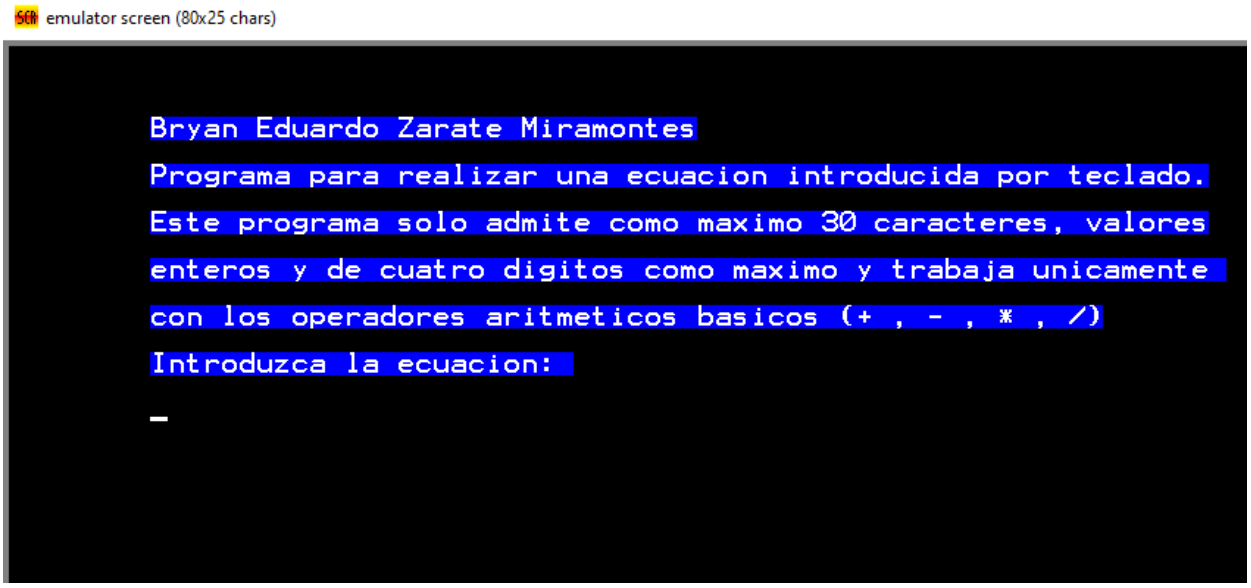
Antecedentes:

A continuación, vamos a crear un programa el cual va a capturar una cadena de caracteres numéricos que representa una ecuación aritmética básica, la cual va a soportar las cuatro operaciones básicas: Suma (+), resta (-), multiplicación (*) y división (/), el programa debe respetar la jerarquía de operaciones.

Además, se debe imprimir en pantalla el nombre del alumno, las instrucciones para poder ingresar la cadena de la ecuación y al final realizará el cálculo de la ecuación introducida para posteriormente imprimir el resultado en pantalla.

Desarrollo

Para la realización de esta práctica, se implementó este código, el cual imprime en pantalla varios mensajes, entre ellos el nombre del alumno (el cual se requiere para la práctica) y las instrucciones del funcionamiento del programa, las cuales le van a indicar al usuario como utilizar dicho programa y que limitantes tiene. Antes de correr el programa se debe de guardar para generar el archivo.asm para posteriormente compilar el programa y así generar tres archivos: el archivo.com, el archivo.list y el archivo.symbol.

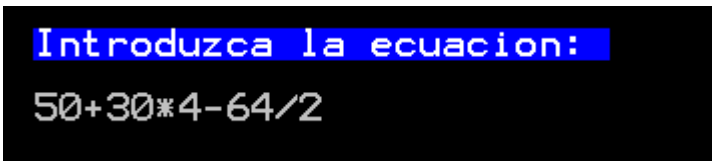


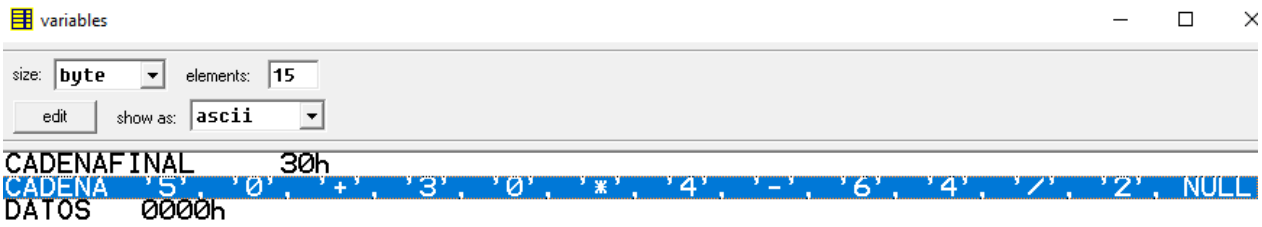
Posterior a la impresión de los mensajes en pantalla continuamos con la ejecución del programa principal el cual consiste en ingresar una cadena de caracteres numéricos que representa una ecuación aritmética básica no mayor a 30 caracteres, en la cual se van a poder realizar sumas, restas, multiplicaciones y divisiones.

Para ello se van a ir leyendo los caracteres que se ingresen por medio del teclado gracias a un bucle que captura cada carácter ingresado por el usuario, detectando la tecla ENTER para finalizar la entrada y BACKSPACE para borrar en caso de errores. Después estos se almacenarán en un arreglo que contendrá toda la cadena ingresada.

```
capturar:
cmp cx, 0x1
jl finCad
mov ah,0
int 16h
cmp al, 0Dh
je finCad
cmp al, 08H
je borrar
mov ah,0Eh

int 10H
mov [di], al
inc di
dec cx
jmp capturar
```

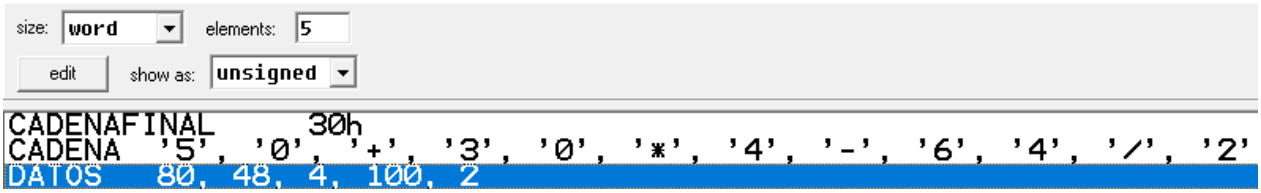
Una vez finalizada la cadena con ENTER, esta se almacena en el arreglo CADENA.



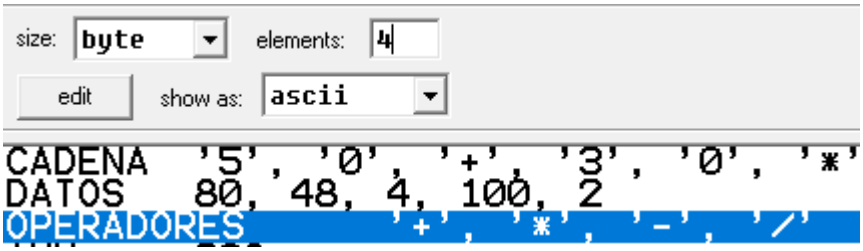


El siguiente paso a realizar es el procesamiento de la cadena, el cual es un proceso que se encarga de transformar los caracteres numéricos de la cadena en valores numéricos hexadecimales y los almacena en la variable DATOS.

```
finCad:
lea di, datos
lea si, cadena
mov cx, 20

siguiente:
xor ax, ax
mov al, [si]
call ascToNum
inc si
cmp al, 0x0
jl finOper
mov [aux], al
xor ax, ax
xor bx, bx
mov ax, [oper]
mov bl, 0x10
mul bx
add al, [aux]
mov [oper], ax
loop siguiente
```

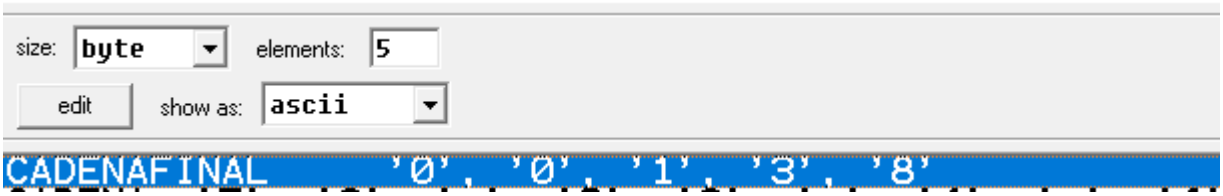


El siguiente paso que realiza el programa es separar los operadores de la cadena ascii e identificar por medio de la jerarquía de operaciones el orden en el cual se van a realizar las operaciones.



Como paso final, el programa realizara las operaciones respetando la jerarquía de operaciones empezando por multiplicaciones y divisiones y al final va a realizar sumas y restas.

Por último, convierte el resultado final de hexadecimal nuevamente a decimal y se va a almacenar en la variable CADENAFINAL la cual se va a imprimir en pantalla.




```
Bryan Eduardo Zarate Miramontes
Programa para realizar una ecuacion introducida por teclado.
Este programa solo admite como maximo 30 caracteres, valores
enteros y de cuatro digitos como maximo y trabaja unicamente
con los operadores aritmeticos basicos (+ , - , * , /)
Introduzca la ecuacion:
50+30*4-64/2
El resultado de la ecuacion es:
00138
```

Código fuente:

```
    name "p6d01"
    org  100h
    jmp msg


;   Este programa obtiene una cadena desde el teclado la cual
;   representa una ecuacion aritmetica utilizando las 4 operaciones
;   aritmeticas basicas que son suma (+) resta (-) multiplicacion (-)
;   y division (/) respectivamente. El programa resuelve esta
;   ecuacion e imprime el resultado en pantalla


;mensajes en pantalla

msg1: db "Bryan Eduardo Zarate Miramontes"
msg2: db "Programa para realizar una ecuacion introducida por teclado."
msg3: db "Este programa solo admite como maximo 30 caracteres, valores"
msg4: db "enteros y de cuatro digitos como maximo y trabaja unicamente
"
msg5: db "con los operadores aritmeticos basicos (+ , - , * , /)"
msg6: db "Introduzca la ecuacion: "
msg7: db "El resultado de la ecuacion es: "
cadenaFinal db "00000", 0
msgerr: db "Ecuacion erronea"
msgend:

msg1_size = msg2 - msg1
msg2_size = msg3 - msg2
msg3_size = msg4 - msg3
msg4_size = msg5 - msg4
msg5_size = msg6 - msg5
msg6_size = msg7 - msg6
msg7_size = cadenaFinal - msg7
cadf_size = msgerr - cadenaFinal
msgerr_size = msgend - msgerr

;variables
cadena      db 20 dup (0)
datos       dw 11 dup (0)
```

```
operadores  db 11 dup (0)
aux         db (0)
oper        dw (0)
corrida     dw 0x0
decimal     dw 11 dup (0)
componente    db ? , ? , ? , ?
num         dw ?
den         db ?
res         dw ?
cos         dw ?
aux2        dw ?
resultado   db ? , ? , ? , ? , ?

;_____________________________________

start:

    mov ah, 0xe
    mov al, 0xa
    int 10H
    int 10H
    mov al, 0xd
    int 10H
    mov al, 0x9
    int 10H
    lea di, cadena
    mov cx, 20

;_____________________________________
capturar:
    cmp cx, 0x1
    jl finCad
    mov ah,0
    int 16h
    cmp al, 0Dh
    je finCad
    cmp al, 08H
    je borrar
    mov ah,0Eh

    int 10H
```

```
    mov [di], al
    inc di
    dec cx
    jmp capturar

borrar:
    cmp cx, 20
    jz capturar
    dec di
    inc cx
    mov ah, 0Eh
    int 10H
    mov al, 0x20
    mov ah, 0x0e
    int 10h
    mov al, 08H
    mov ah, 0Eh
    int 10H
    jmp capturar

finCad:
    lea di, datos
    lea si, cadena
    mov cx, 20

siguiente:
    xor ax, ax
    mov al, [si]
    call ascToNum
    inc si
    cmp al, 0x0
    jl finOper
    mov [aux], al
    xor ax, ax
    xor bx, bx
    mov ax, [oper]
    mov bl, 0x10
    mul bx
    add al, [aux]
    mov [oper], ax
    loop siguiente
```

```
finOper:
    inc corrida
    mov [aux], al
    xor ax, ax
    mov ax, [oper]
    mov [di], ax
    inc di
    inc di
    cmp [aux], -48
    jz finOperandos
    mov [oper], 0x0
    mov [aux], 0x0
    cmp cx, 0x0
    jz finOperandos
    loop siguiente

finOperandos:
    lea di, operadores
    lea si, cadena
    mov cx, 20

operators:
    xor ax, ax
    mov al, [si]
    inc si
    cmp al, '*'
    je pilaOperadores
    cmp al, '/'
    je pilaOperadores
    cmp al, '+'
    je pilaOperadores
    cmp al, '-'
    je pilaOperadores
    loop operators

pilaOperadores:
    mov [di], al
    inc di
    cmp cx, 0x0
```

```
    jz cad
    loop operators

cad:
    lea di, decimal
    lea si, datos
    mov cx, [corrida]
    mov [den], 0x10


decToHex:
    mov ax, [si]
    mov [num], ax
    inc si
    inc si
    pusha
    lea di, componente+3

retry:
    call division
    xor ax, ax
    xor bx, bx
    xor dx, dx
    mov ax, [res]
    mov [di], al
    dec di
    xor ax, ax
    mov ax, [cos]
    mov [num], ax
    mov bl, [den]
    cmp ax, bx
    jl finHex
    jmp retry

finHex:
    mov [di], al
    lea si, componente
    xor ax, ax
    xor bx, bx
    mov al, [si]
    mov bx, 1000
```

```
mul bx
mov [aux2], ax
inc si
xor ax, ax
xor bx, bx
mov al, [si]
mov bl, 100
mul bx
xor bx, bx
mov bx, [aux2]
adc bx, ax
mov [aux2], bx
inc si
xor ax, ax
xor bx, bx
mov al, [si]

mov bl, 10
mul bx
xor bx, bx
mov bx, [aux2]
adc bx, ax
mov [aux2], bx
inc si

xor ax, ax
xor bx, bx
mov ax, [aux2]
mov bl, [si]
adc ax, bx
mov [aux2], ax

call limpiarArreglo
popa
mov ax, [aux2]
mov [di], ax
inc di
inc di
loop decToHex
lea si, decimal
lea di, operadores
```

```
    mov cx, [corrida]

;_____________________________________
dm:
    mov al, [di]
    cmp al, '*'
    je multiplica
    cmp al, '/'
    je divide
    cmp cx, 0x0
    jz findm
    inc di
    inc si
    inc si
    loop dm
    cmp cx, 0x0
    jz findm


multiplica:
    xor ax, ax
    xor bx, bx
    mov ax, [si]
    inc si
    inc si
    mov bx, [si]
    mul bx
    dec si
    dec si
    mov [si],ax
    inc si
    inc si
    call refrescar
    dec si
    dec si
    call refOpers
    loop dm

divide:
    xor ax, ax
    xor dx, dx
```

```
    xor bx, bx
    mov ax, [si]
    inc si
    inc si
    mov bx, [si]
    div bx
    dec si
    dec si
    mov [si], ax
    inc si
    inc si
    call refrescar
    dec si
    dec si
    call refOpers
    loop dm

findm:
    lea si, decimal
    lea di, operadores
    mov cx, [corrida]


sumRes:
    mov al, [di]
    cmp al, '+'
    je suma
    cmp al, '-'
    je resta
    cmp cx, 0x0
    jz conversor:
    inc di
    inc si
    inc si
    loop sumRes
    cmp cx, 0x0
    jz conversor:

suma:
    xor ax, ax
    xor bx, bx
```

```
    mov ax, [si]
    inc si
    inc si
    mov bx, [si]
    add ax, bx
    dec si
    dec si
    mov [si], ax
    inc si
    inc si
    call refrescar
    dec si
    dec si
    call refOpers
    loop sumRes

resta:
    xor ax, ax
    xor dx, dx
    xor bx, bx
    mov ax, [si]
    inc si
    inc si
    mov bx, [si]
    sub ax, bx
    dec si
    dec si
    mov [si],ax
    inc si
    inc si
    call refrescar
    dec si

    dec si
    call refOpers
    loop sumRes

refOpers:
    pusha
    mov cx, 11
```

```
retryOp:
    mov ax, [di+1]
    mov [di], ax
    cmp ax, 0x0
    jz finRetry
    inc di
    loop retryOp

finRetry:
    popa
    ret

refrescar:
    pusha
    mov cx, 11

repet:
    mov ax, [si+2]
    mov [si], ax
    cmp ax, 0x0
    jz finRep
    inc si
    inc si
    loop repet

finRep:
    popa
    ret


conversor:
    xor ax, ax
    xor bx, bx
    lea si, decimal
    lea di, resultado+4
    mov [den], 0x0a
    mov ax, [si]
    mov [num], ax

convertirDecimal:
    call division
```

```
    xor ax, ax
    mov al, b.[res]
    mov [di], al
    dec di
    xor bx, bx
    mov ax, [cos]
    mov [num], ax
    xor bx, bx
    mov bl, [den]
    cmp ax, bx
    jle finDecimal
    jmp convertirDecimal

finDecimal:
    mov [di], al
    xor cx, cx
    mov cl, 0x5
    lea si, resultado
    lea di, cadenaFinal

copiar:
    mov al, [si]
    add [di], al
    inc si
    inc di
    loop copiar
    jmp fin

division:
    pusha
    xor ax, ax
    xor bx, bx
    mov ax,[num]
    mov bl, [den]

    div bx
    mov [cos], ax
    mov [res], dx
    popa
    ret
```

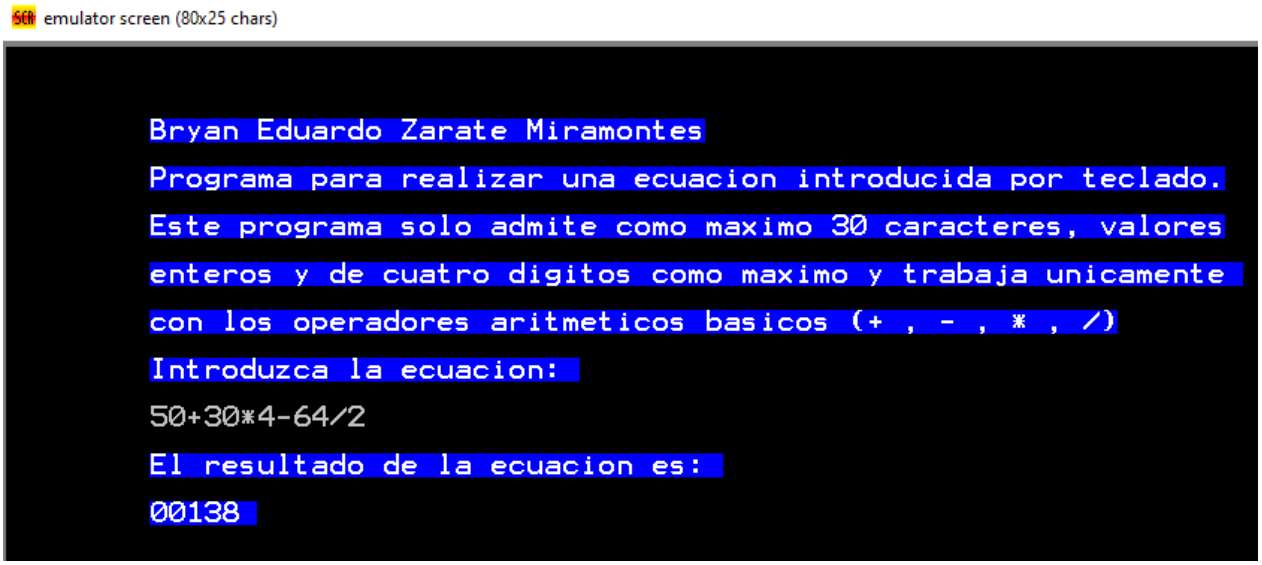
```
limpiarArreglo:
    pusha
    lea si, componente
    mov [si], 0x0
    inc si
    mov [si], 0x0
    inc si
    mov [si], 0x0
    inc si
    mov [si], 0x0
    inc si
    popa
    ret

ascToNum:
    sub al, 48
    cmp al, 0x9
    jnle error
    ret

error:
    mov al, 1
    mov bh, 0
    mov dl, 3
    mov dh, 10
    mov cx, msgerr_size
    mov bp, offset msgerr
    mov ah, 13h
    int 10h

msg:
    mov dx, 0308h
    mov bx, 0
    mov bl, 10011111b
    mov cx, msg1_size
    mov al, 01b
    mov bp, msg1
    mov ah, 13h
    int 10h

    mov dx, 0508h
```

```
mov bx, 0
mov bl, 10011111b
mov cx, msg2_size
mov al, 01b
mov bp, msg2
mov ah, 13h
int 10h

mov dx, 0708h
mov bx, 0
mov bl, 10011111b
mov cx, msg3_size
mov al, 01b
mov bp, msg3
mov ah, 13h
int 10h

mov dx, 0908h
mov bx, 0
mov bl, 10011111b
mov cx, msg4_size
mov al, 01b
mov bp, msg4
mov ah, 13h
int 10h

mov dx, 0B08h
mov bx, 0
mov bl, 10011111b
mov cx, msg5_size
mov al, 01b
mov bp, msg5
mov ah, 13h
int 10h

mov dx, 0D08h
mov bx, 0
mov bl, 10011111b
mov cx, msg6_size
mov al, 01b
mov bp, msg6
```

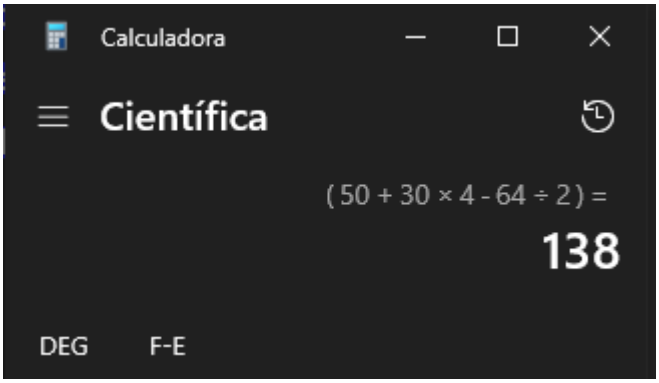
```
    mov ah, 13h
    int 10h

    jmp start

fin:
    mov dx, 1108h
    mov bx, 0
    mov bl, 10011111b
    mov cx, msg7_size
    mov al, 01b
    mov bp, msg7
    mov ah, 13h
    int 10h

;;;;;;;;;;;;;;;;; 12

    mov dx, 1308h
    mov cx, cadf_size
    mov bp, offset cadenaFinal
    mov ah, 13h
    int 10h
    int 20h
```

Resultados obtenidos

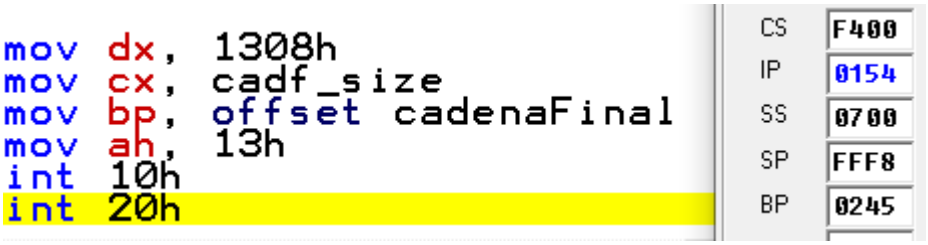
Al finalizar el programa, nos mostrara este mensaje en pantalla:



Para la comprobación del resultado, realizamos la sumatoria en la calculadora y el resultado fue el esperado



Al ejecutar la interrupción 20h el programa finaliza y el registro SP almacena el valor FFF8, lo que indica que el programa finalizo de manera correcta.



Conclusiones

El programa presentado permite ingresar una ecuación aritmética simple desde el teclado y procesa las operaciones aritméticas básicas, mostrando el resultado final en pantalla. La implementación es detallada y compleja, manejando correctamente la entrada del usuario y realizando conversiones necesarias entre representaciones numéricas. Aunque funcional, la lógica podría mejorarse para mayor eficiencia y manejo de errores.