

# UNIVERSIDAD DE GUADALAJARA

## CENTRO UNIVERSITARIO DE CIENCIAS SOCIALES E INGENIERÍA



### Proyecto - Compilador

**Alumno:** Alan David Vélez Gómez

**Código:** 216804649

**Maestro:** Michel Emanuel López Franco

**Carrera:** Ingeniería en computación

**Materia:** Seminario de solución de problemas de traductores de lenguaje 2, D02

Al ejecutar el programa podemos ver la interfaz inicial, en la caja de texto de la cadena de entrada se ingresará el código.

The screenshot shows the 'Analizador Lexico' application window. It has a teal background and contains several components:

- Cadena de entrada:** A large white text area on the top left for entering code.
- Validar:** A button to the right of the input area.
- Conversión a ensamblador:** A large white text area on the top right for assembly output.
- Análisis sintáctico:** A button below the input area.
- Análisis semántico:** A button below the assembly output area.
- Table:** A table with three columns: 'Entrada', 'Tipo', and 'Num'. The first row contains an asterisk (\*).
- Limpiar:** A button at the bottom center.

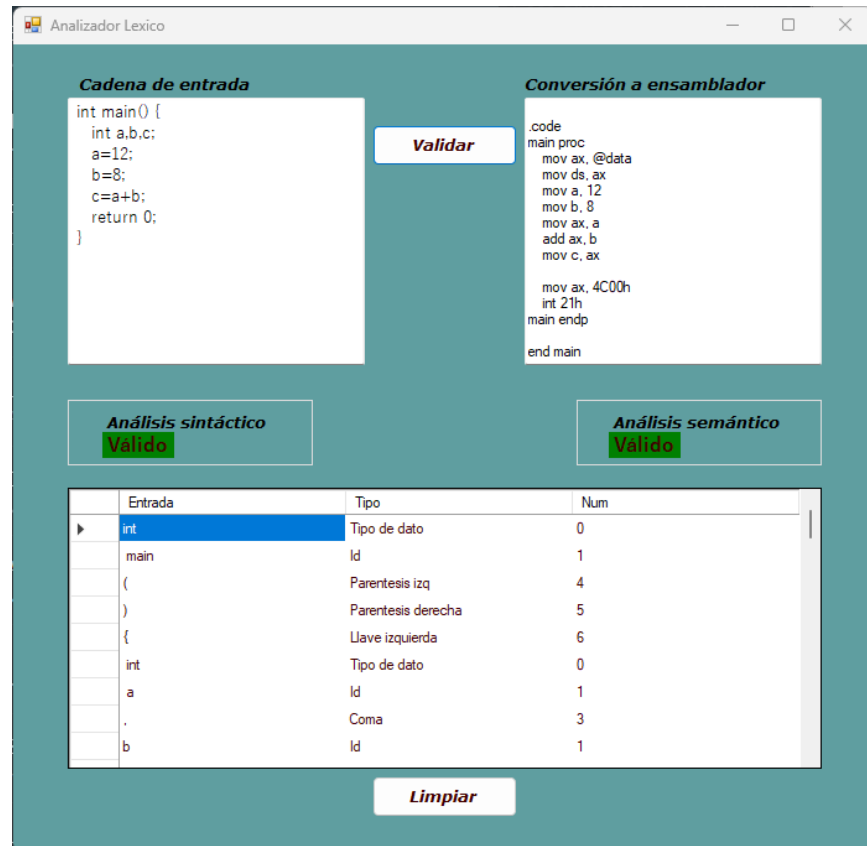
Ingresamos el código y posteriormente debemos oprimir el botón de “Validar”.

This screenshot shows the same application window after the code has been entered. The 'Cadena de entrada' field now contains the following C code:

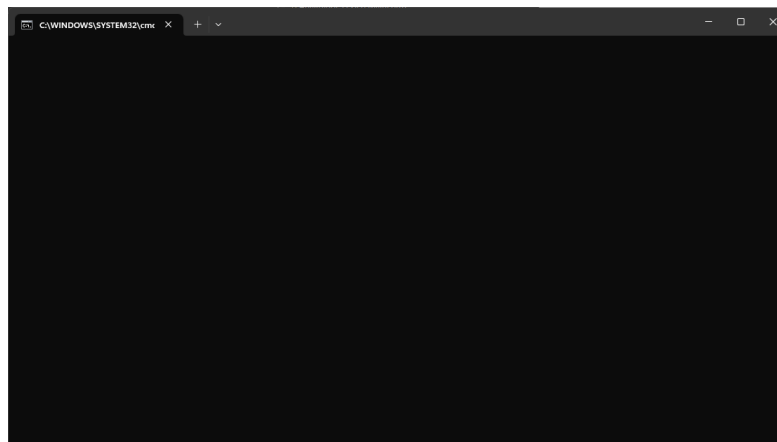
```
int main() {  
    int a,b,c;  
    a=12;  
    b=8;  
    c=a+b;  
    return 0;  
}
```

The 'Validar' button remains visible next to the input area. The rest of the interface, including the 'Conversión a ensamblador' area, the analysis buttons, the table, and the 'Limpiar' button, remains the same as in the previous screenshot.

Con el botón nos realiza todos los análisis que pedimos, primeramente, nos crea la tabla con los tokens, los tipos de datos y los números correspondientes al tipo. Una vez completo el análisis léxico, pasa al análisis sintáctico y semántico, en este se crea el árbol sintáctico y la pila donde se almacenan los valores para validar. Con las reglas se va validando cada token y al final si llega al estado de aceptación nos muestra que análisis es válido. Por último nos convierte a ensamblador parte de nuestro código, esto gracias al árbol sintáctico que separa los tokens.



Al mismo tiempo crea un archivo .exe en el cual debería escribirse el resultado de la operación, pero como no estamos aceptando "printf", "cout" o alguna instrucción de impresión, simplemente nos imprime el cmd vacío por la misma razón.



En este punto mi código hace que se abra un archivo en .asm con el código generado solo como comprobación, en esta comprobación se puede ver que el código compila perfectamente en el emu8086 y hace la suma dando el resultado correcto.

The image shows the emu8086 IDE with the assembly source code in the main window and the emulator running it in the bottom window. The source code is as follows:

```
01 .model small
02 .stack 100h
03
04 .data
05     a dw 0
06     b dw 0
07     c dw 0
08
09 .code
10 main proc
11     mov ax, @data
12     mov ds, ax
13     mov a, 12
14     mov b, 8
15     mov ax, a
16     add ax, b
17     mov c, ax
18
19     mov ax, 4C00h
20     int 21h
21 main endp
22
23 end main
```

The emulator window shows the registers and the execution of the code. The registers are as follows:

Register	Value
AX	0014
BX	0000
CX	0130
DX	0000
SI	0018
DI	0000
SP	0710
BP	0000
SI	0000
DI	0000
DS	0720
ES	0700

The execution log shows the following instructions being executed:

```
07221: A1 161 I MOV AX, [00000h]
07222: 00 000 NI ADD AX, [00002h]
07223: 00 000 NI MOV [00004h], AX
07224: 03 003 MOV AX, 04C00h
07225: 05 006 INT 021h
07226: 02 002 NOP
07227: 00 000 NI NOP
07228: A3 163 C NOP
07229: 04 004 NOP
0722A: 00 000 NI NOP
0722B: B8 184 @ NOP
0722C: 00 000 NI NOP
0722D: 4C 076 L NOP
0722E: 00 205 NOP
0722F: 21 033 NOP
07230: 90 144 NOP
07231: 90 144 E ...
```

Ahora un caso no válido, en este caso es la misma cadena, pero borré la declaración de la “b”, lo cual es válido léxico y sintácticamente, pero en el semántico ya nos marca el error, por lo que tampoco puede traducirse al ensamblador.

**Cadena de entrada**

```
int main() {
  int a,c;
  a=12;
  b=8;
  c=a+b;
  return 0;
}
```

**Validar**

**Conversión a ensamblador**  
No se puede convertir

**Análisis sintáctico**  
**Valido**

**Análisis semántico**  
**NO válido**

	Entrada	Tipo	Num
▶	int	Tipo de dato	0
	main	Id	1
	(	Parentesis izq	4
	)	Parentesis derecha	5
	{	Llave izquierda	6
	int	Tipo de dato	0
	a	Id	1
	,	Coma	3
	b	Id	1

**Limpiar**

Otro caso inválido, en este caso quité un “;” y agregué una “s” en un lugar al azar para que me diera error y después de hacer el análisis léxico, nos marca que no es válido sintácticamente.

**Cadena de entrada**

```
int main() {
  int a,b,c;
  a=12
  b=8;
  c=a+b: s
  return 0;
}
```

**Validar**

**Conversión a ensamblador**  
No se puede convertir

**Análisis sintáctico**  
**NO válido**

**Análisis semántico**  
**NO válido**

	Entrada	Tipo	Num
▶	int	Tipo de dato	0
	main	Id	1
	(	Parentesis izq	4
	)	Parentesis derecha	5
	{	Llave izquierda	6
	int	Tipo de dato	0
	a	Id	1
	,	Coma	3
	b	Id	1

**Limpiar**