Compilers

C O M P I L A D O R

Team 04.

INTRODUCTION

Problem solved

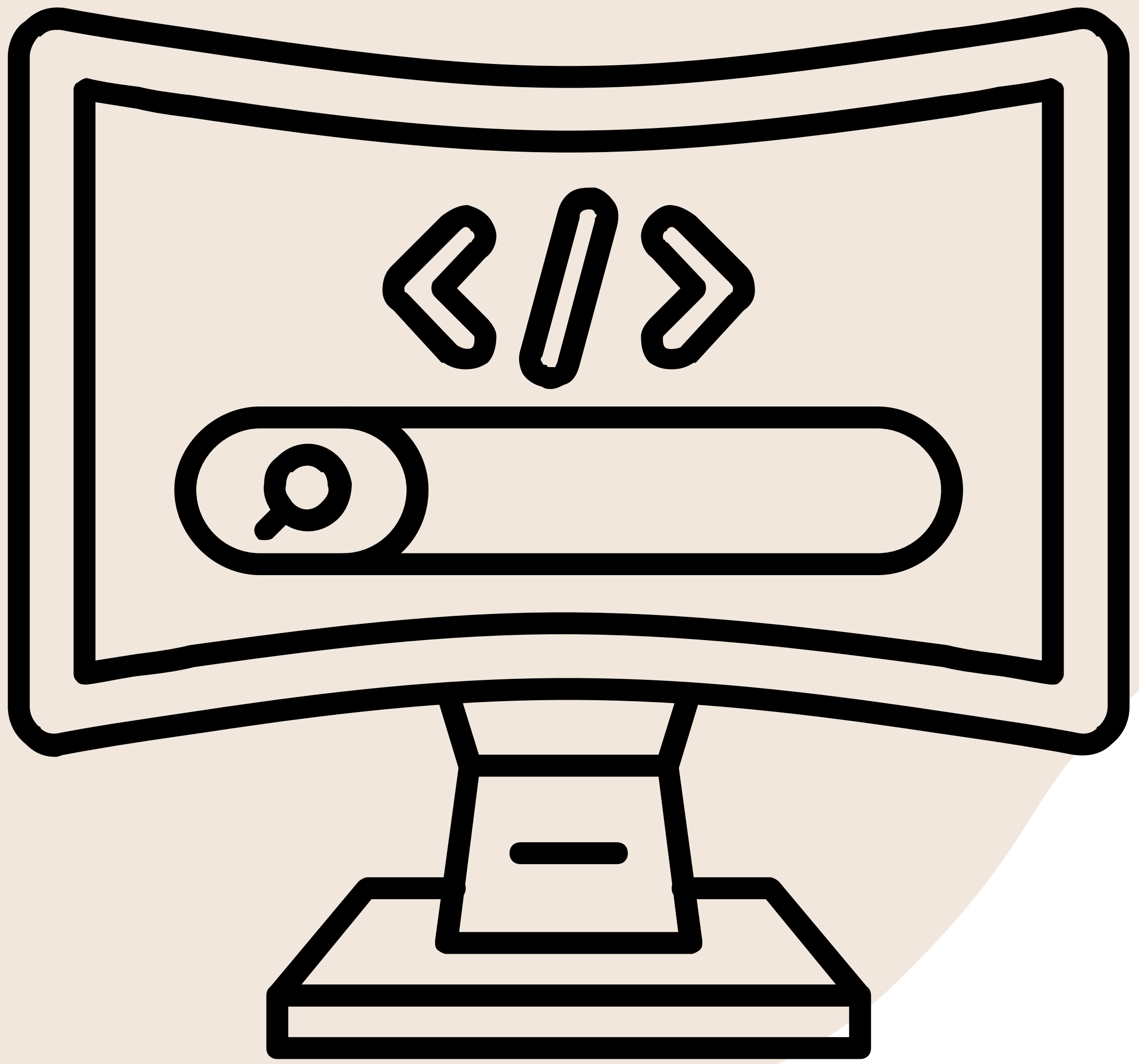
This project addresses a fundamental compiler problem.
Given a sequence of tokens we verify if the program is syntactically correct, semantically valid, and then translate it.

Objective

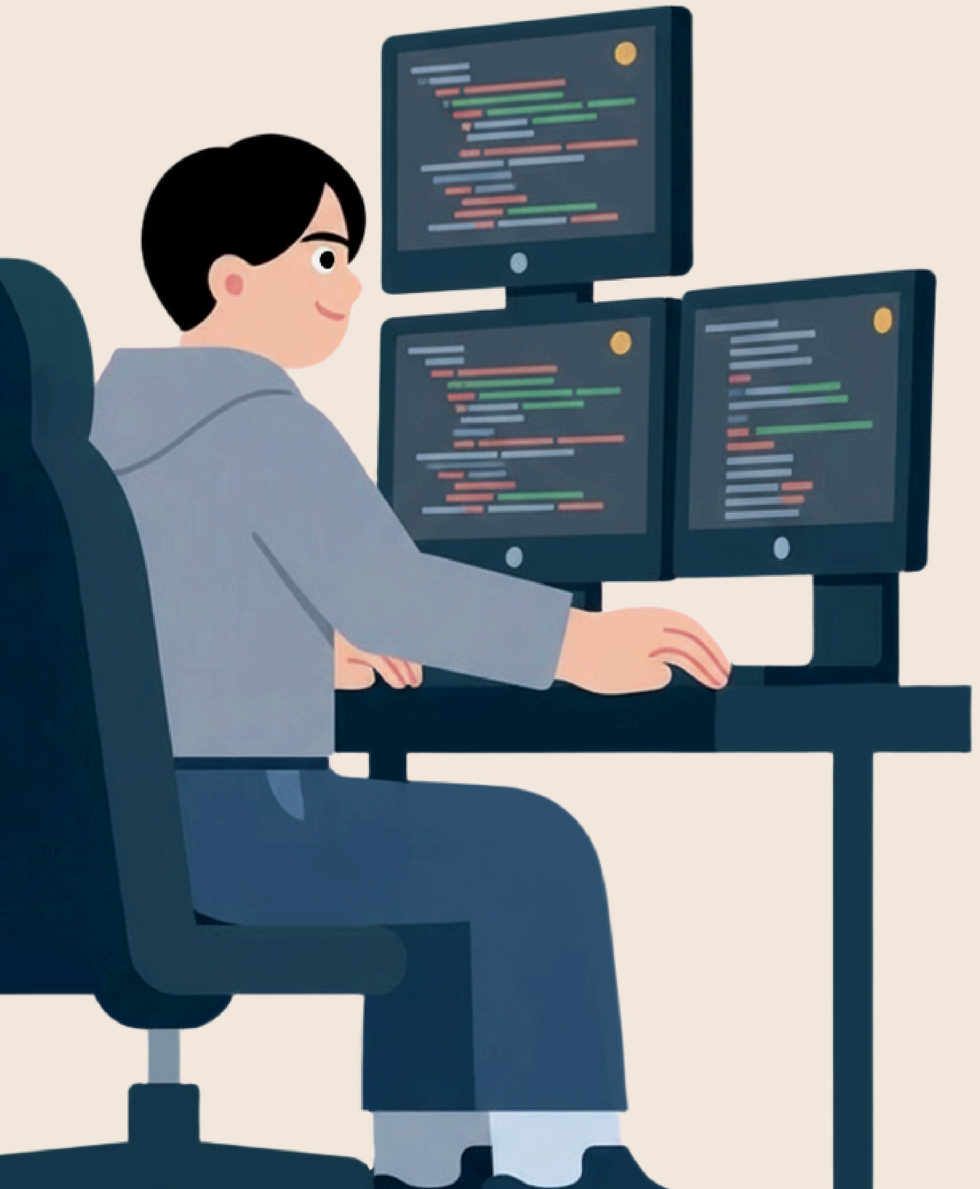
- Design a parser
- Introduce semantic verification
- Generate Three-Address Code (TAC)
- Compile and execute
- Provide clear feedback

C supported

The project implements a reduced sub-language of C



GENERAL ARCHITECTURE



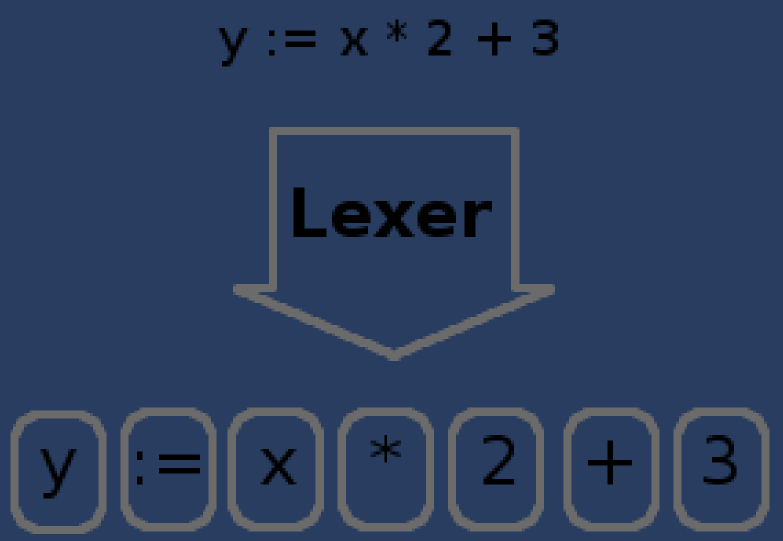
LEXER
(Tokens)

PARSER + SDT
(Syntax Semantics)

TAC
(Three Address Code)

VM / ASM
(Execution)

LEXER



01

The lexer is the first phase of the compiler.

It reads the input file character by character and groups them into tokens.

Removes whitespace, newlines, and comments.

02

Recognizes:

- Identifiers
- Numbers
- Keywords
- Operators
- Delimiters
- Produces a linear token stream used by the parser.

03

EXAMPLE

The line int x = 10; translates to:

Keyword Identifier Operator Constant Punctuation.

Tokens produced:

[int] [x] [=] [10] [;]

PARSER

- The parser receives the token stream from the lexer.
- Its job is to check whether the sequence follows the grammar of the C subset.
- We implemented a Bottom-Up parser, based on:
  - An analysis stack
  - The try_reduce() function to apply grammar rules
- Detects syntax errors, such as:
  - Misordered tokens
  - Incomplete structures
  - Invalid declarations

EXAMPLE

Tokens:
int x = 10 ;

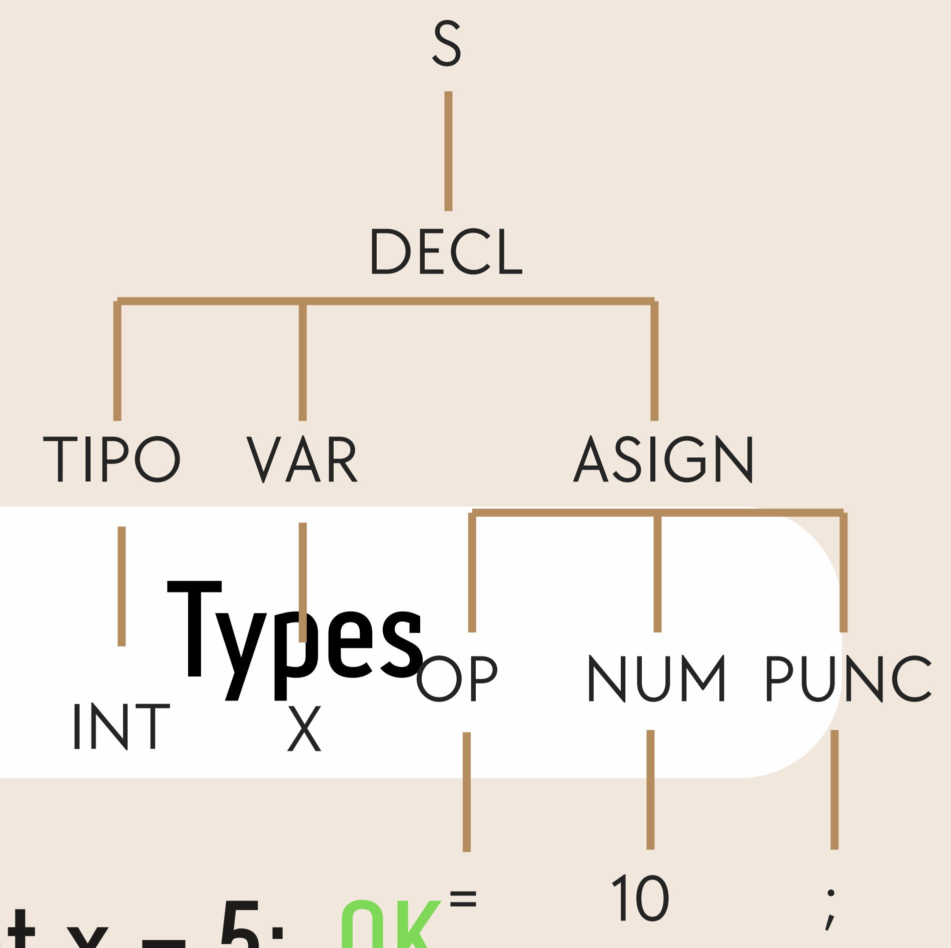
Reductions:
int → TYPE
x → ID
TYPE ID = NUM ; → DECLARATION

SEMANTIC ANALYSIS (SDT)

ERRORS

- Parsing Error (Syntax)
  - int x = ; missing item
- SDT Error (Semantic):
  - int x;
  - x = "cadena"; // tipos incompatibles

Parser Tree



Types

- int x = 5; OK
- int x = "hola"; error

Grammar

(1) S → DECL
(2) DECL → TIPO VAR ASIGN
(3) TIPO → KW
(4) VAR → ID
(5) ASIGN → OP NUM PUNC
(5) ASIGN → OP LIT PUNC

INTERMEDIATE CODE (THREE ADDRESS CODE – TAC)

Central idea: each operation is broken down into simple instructions with a maximum of three operands (destination = op1 operator op2).

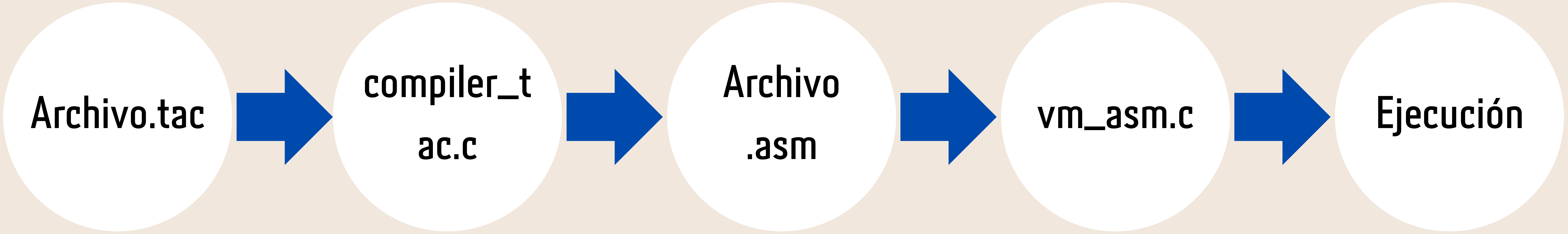
Advantage:

- Machine independence → easier to optimize.
- Clarity in flow control (labels, jumps).
- Basis for translating to ASM.

EXAMPLE

```
x = a + b * c;

t1 = b * c
x  = a + t1
```

INTERMEDIATE CODE TO MACHINE



Example Code (Left Side TAC vs Right Side ASM):

```
TAC: t1 = a+ b

ASM: LOAD A, a
ADD A, b
STORE A, t1
```

THE VIRTUAL MACHINE (EXECUTION)

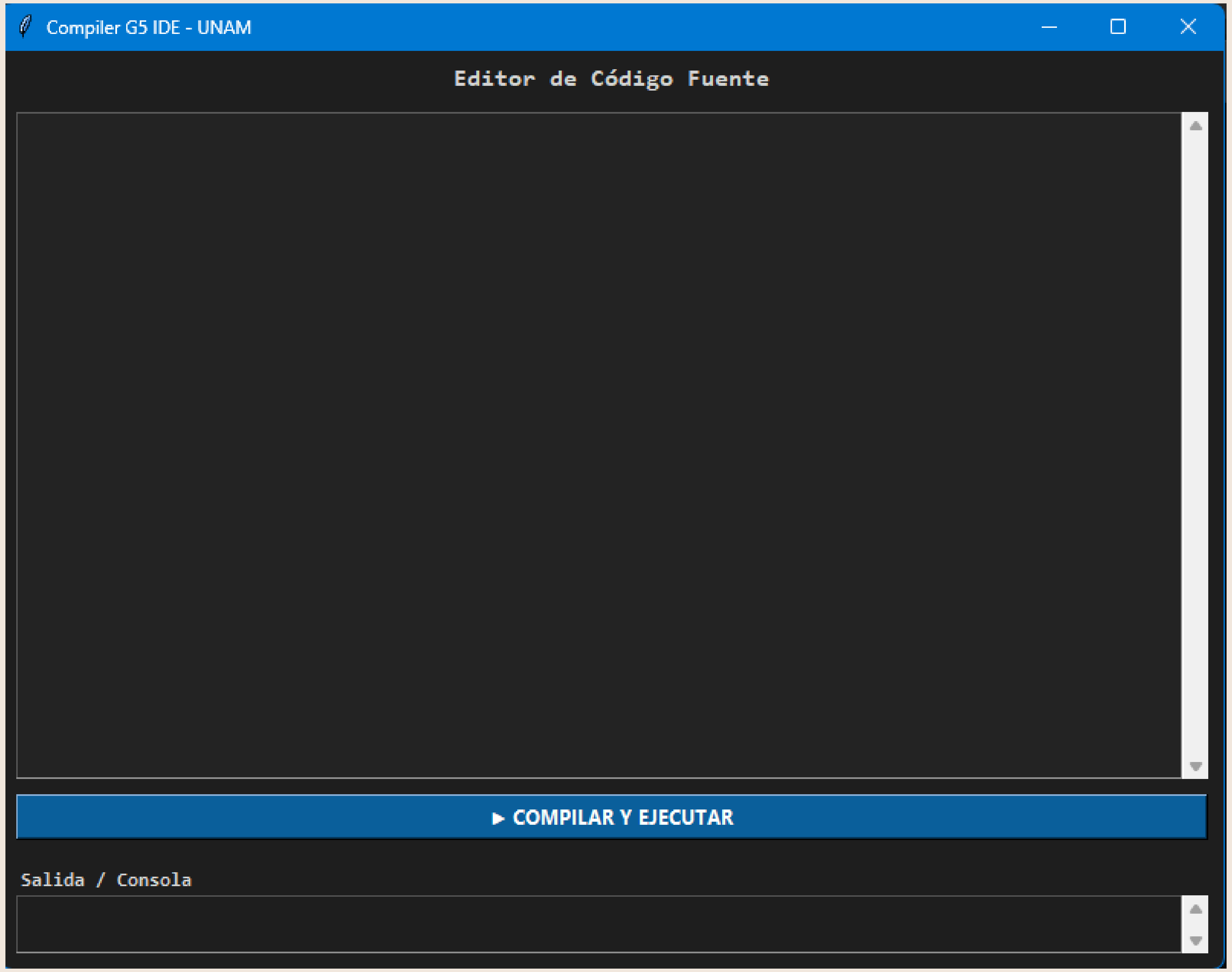
List of VM Features:

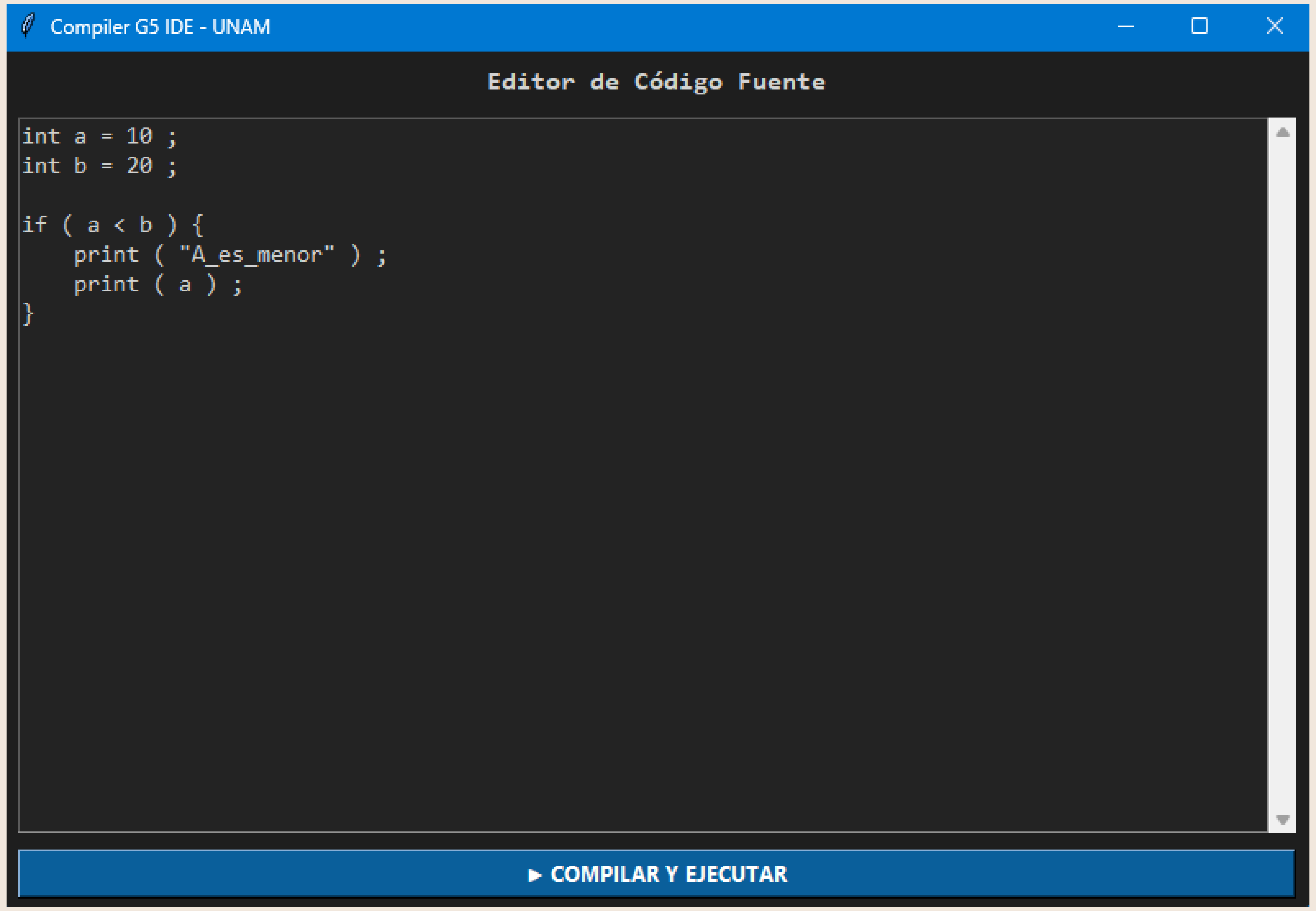
- Accumulator Architecture (Register 'A').
- Data Memory (Dynamic Symbol Table).
- Supported Instructions: Arithmetic (ADD, SUB), Logic (JGT, JLT), I/O (PRINT).

Code snippet from vm_asm.c:

```
case  OP_ADD:
      A += get_value(inst->arg);
      pc++;
```

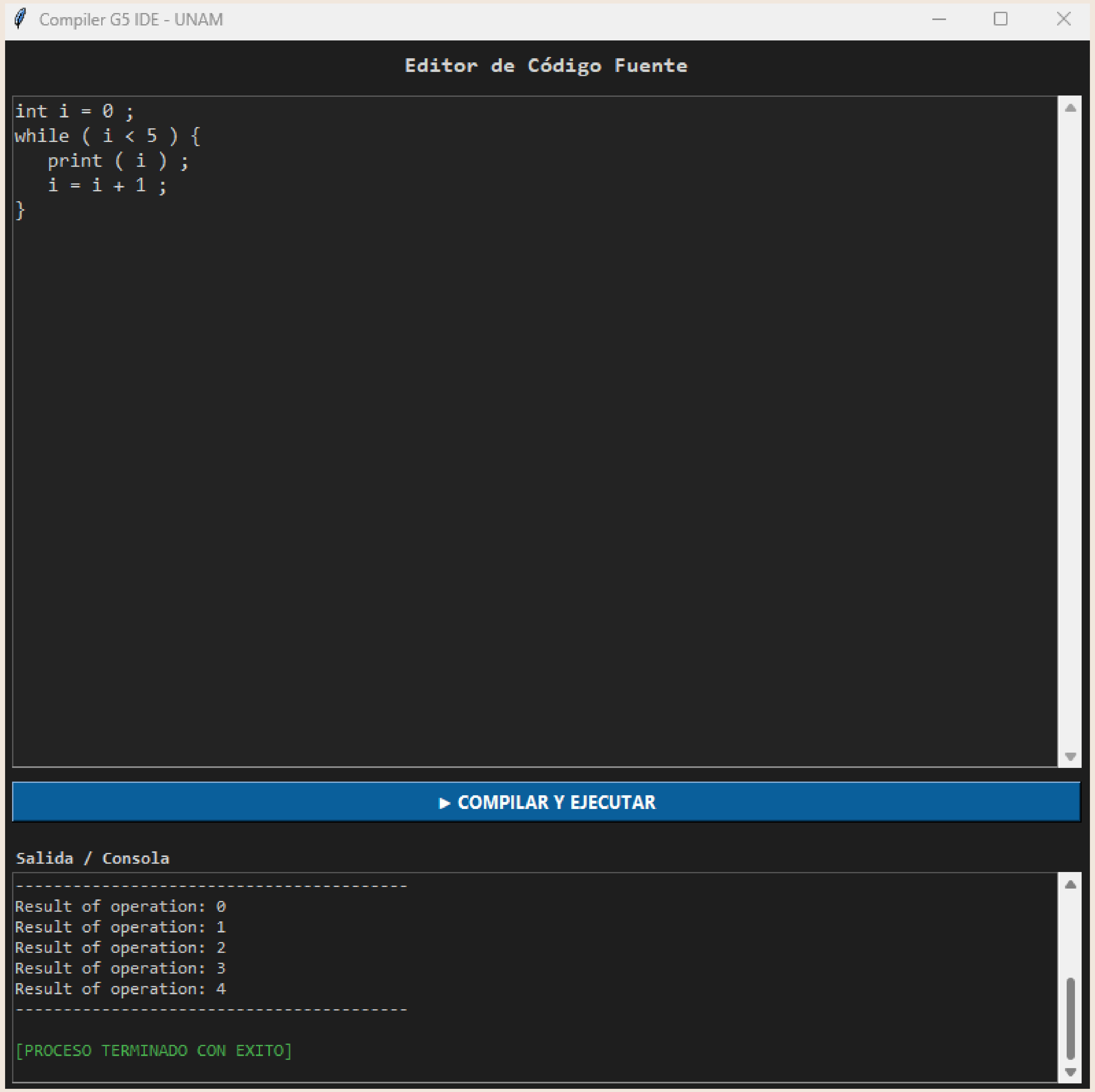
GRAPHICAL USER INTERFACE





```
int a = 10 ;
int b = 20 ;

if ( a < b ) {
    print ( "A_es_menor" ) ;
    print ( a ) ;
}
```

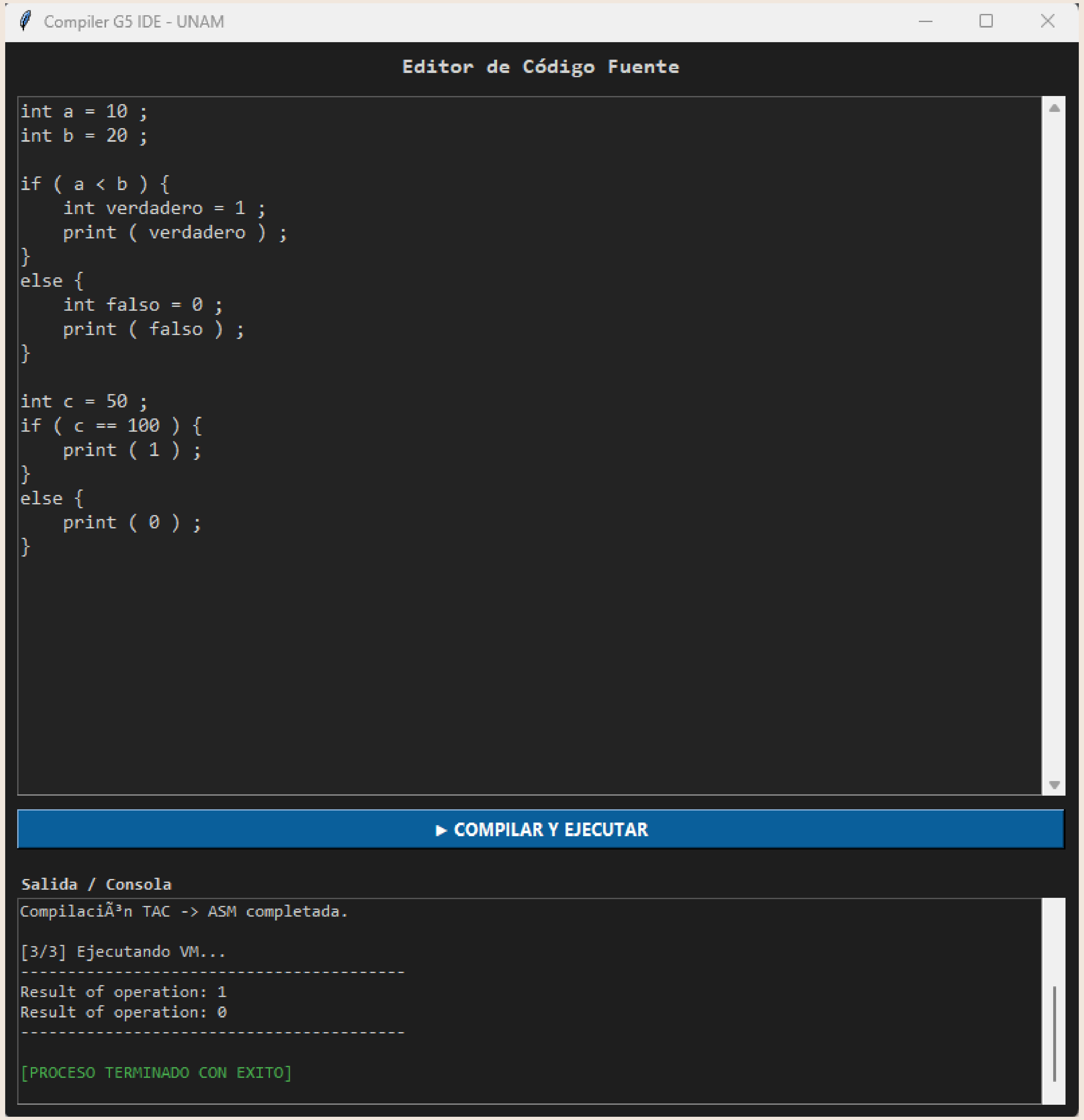
SUCCESSFULLY COMPLETED (WHILE)



```
int i = 0 ;
while ( i < 5 ) {
   print ( i ) ;
   i = i + 1 ;
}
```

```
----------------------------------------
Result of operation: 0
Result of operation: 1
Result of operation: 2
Result of operation: 3
Result of operation: 4
----------------------------------------

[PROCESO TERMINADO CON EXITO]
```

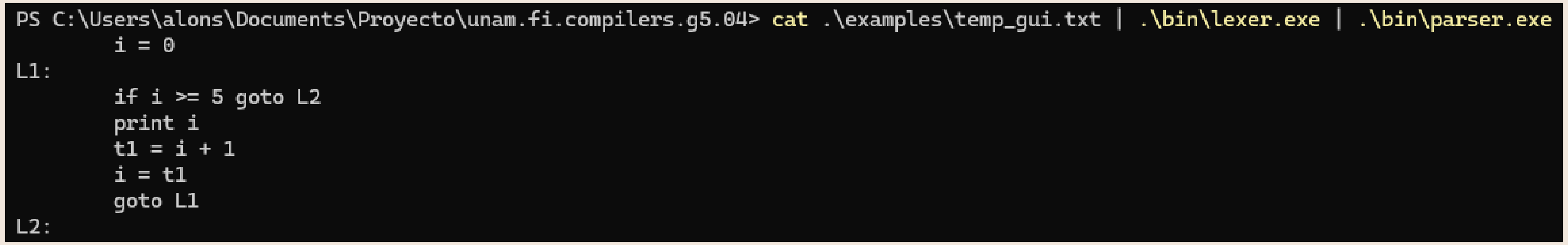
SUCCESSFULLY COMPLETED(IF ELSE)



```
int a = 10 ;
int b = 20 ;

if ( a < b ) {
    int verdadero = 1 ;
    print ( verdadero ) ;
}
else {
    int falso = 0 ;
    print ( falso ) ;
}

int c = 50 ;
if ( c == 100 ) {
    print ( 1 ) ;
}
else {
    print ( 0 ) ;
}
```

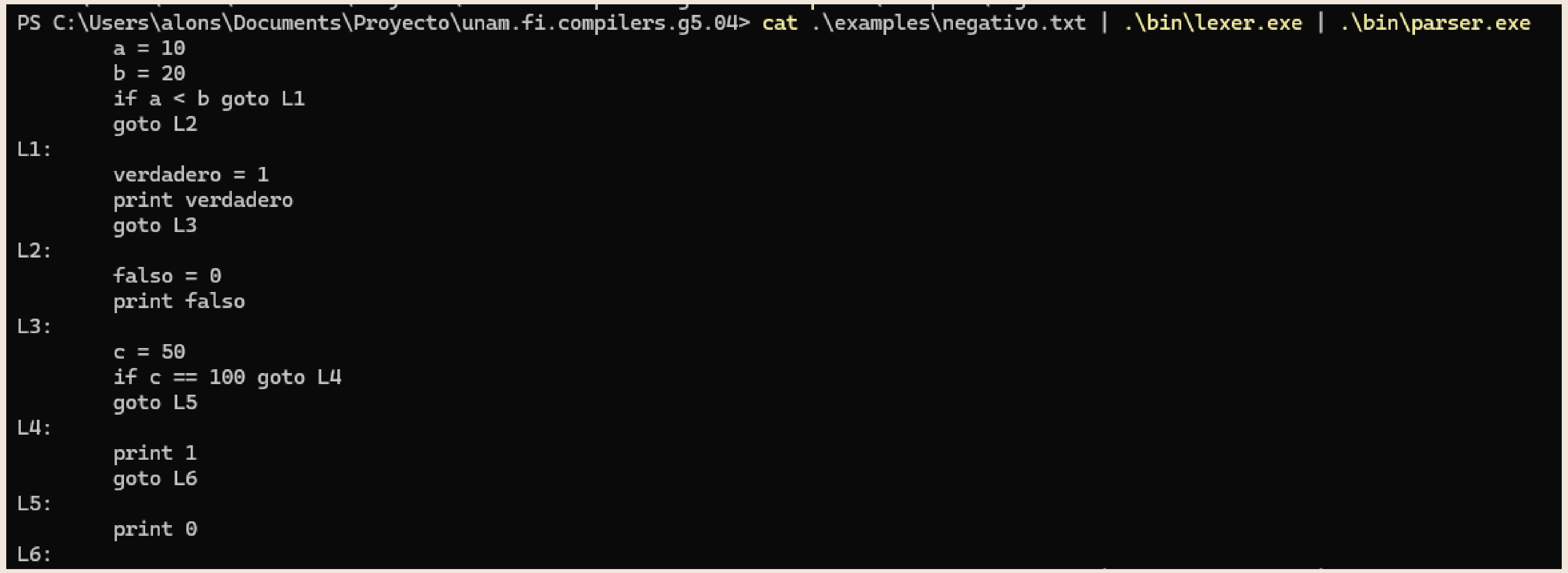
```
CompilaciÃ³n TAC -> ASM completada.

[3/3] Ejecutando VM...
----------------------------------------
Result of operation: 1
Result of operation: 0
----------------------------------------

[PROCESO TERMINADO CON EXITO]
```

TAC FROM THE LAST EXAMPLE (WHILE)



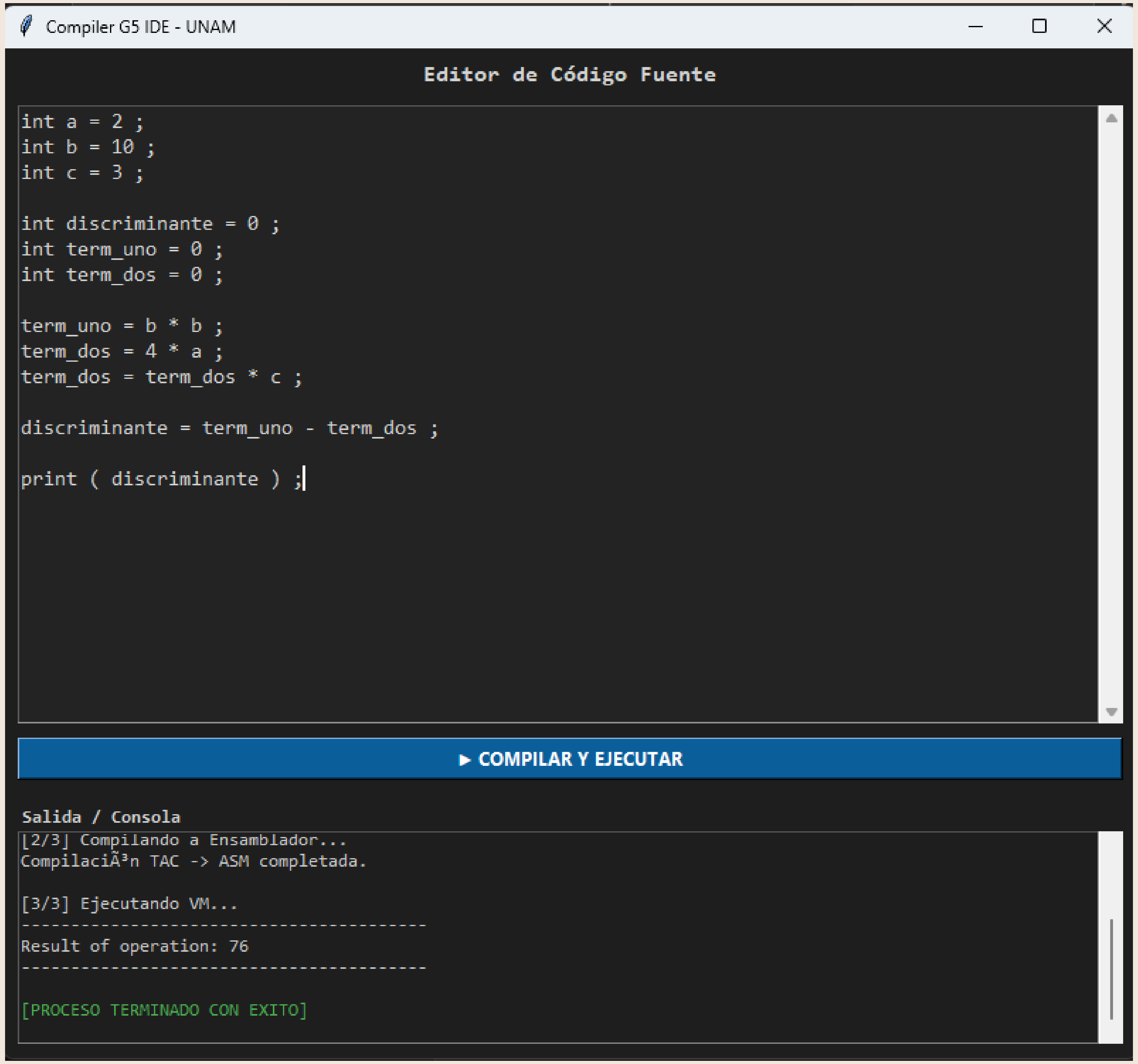
```
PS C:\Users\alons\Documents\Proyecto\unam.fi.compilers.g5.04> cat .\examples\temp_gui.txt | .\bin\lexer.exe | .\bin\parser.exe
        i = 0
L1:
        if i >= 5 goto L2
        print i
        t1 = i + 1
        i = t1
        goto L1
L2:
```

An other example (IF ELSE)



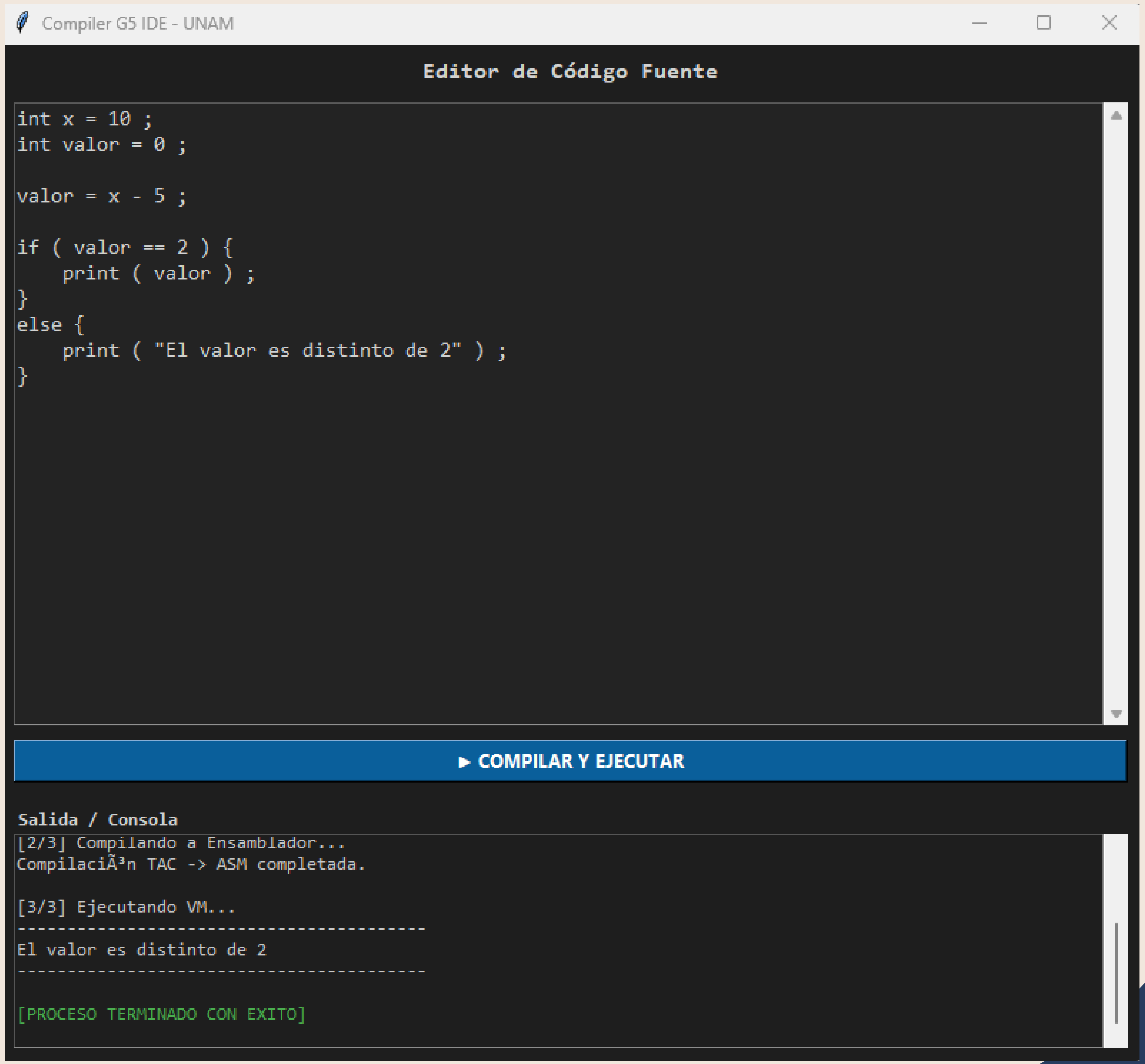
```
PS C:\Users\alons\Documents\Proyecto\unam.fi.compilers.g5.04> cat .\examples\negativo.txt | .\bin\lexer.exe | .\bin\parser.exe
        a = 10
        b = 20
        if a < b goto L1
        goto L2
L1:
        verdadero = 1
        print verdadero
        goto L3
L2:
        falso = 0
        print falso
L3:
        c = 50
        if c == 100 goto L4
        goto L5
L4:
        print 1
        goto L6
L5:
        print 0
L6:
```

SUCCESSFULLY COMPLETED



```
int a = 2 ;
int b = 10 ;
int c = 3 ;

int discriminante = 0 ;
int term_uno = 0 ;
int term_dos = 0 ;

term_uno = b * b ;
term_dos = 4 * a ;
term_dos = term_dos * c ;

discriminante = term_uno - term_dos ;

print ( discriminante ) ;
```

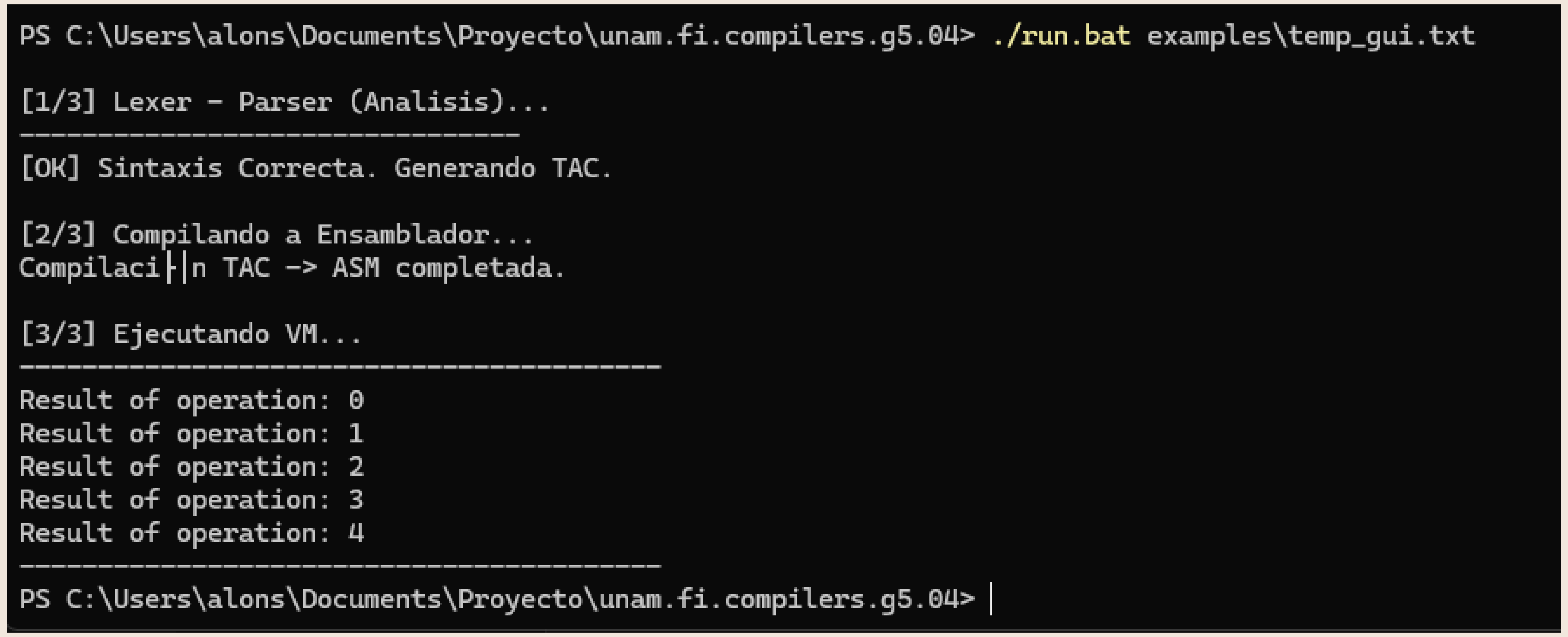
```
[2/3] Compilando a Ensamblador...
CompilaciÃ³n TAC -> ASM completada.

[3/3] Ejecutando VM...
------------------------------------------
Result of operation: 76
------------------------------------------

[PROCESO TERMINADO CON EXITO]
```



```
int x = 10 ;
int valor = 0 ;

valor = x - 5 ;

if ( valor == 2 ) {
    print ( valor ) ;
}
else {
    print ( "El valor es distinto de 2" ) ;
}
```

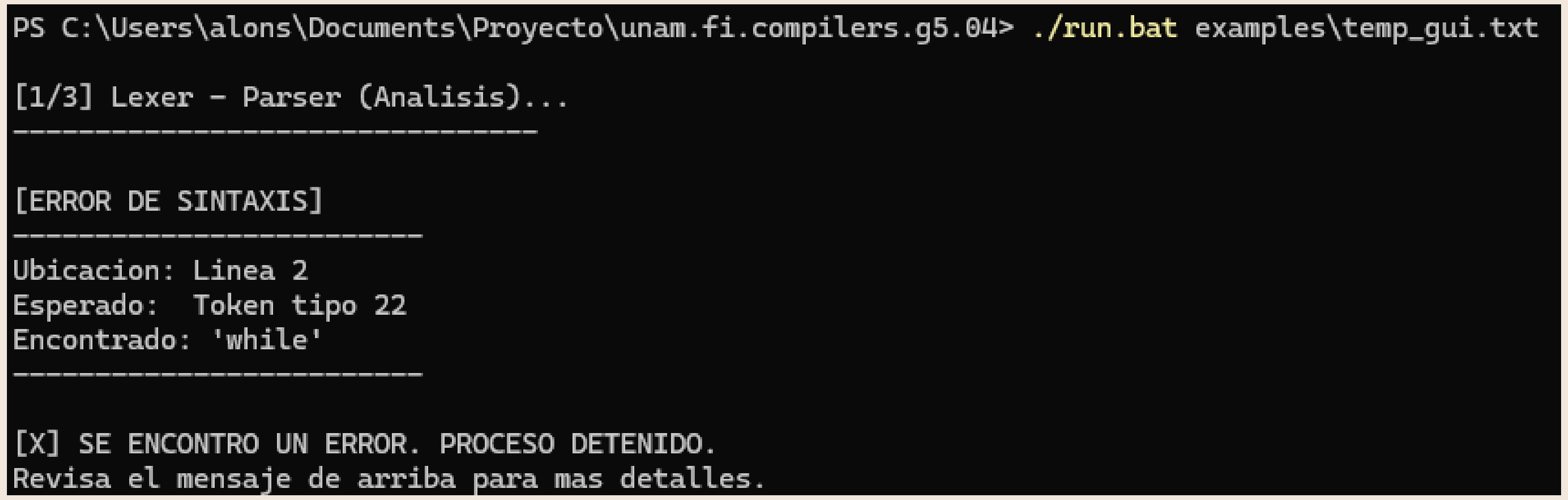
```
[2/3] Compilando a Ensamblador...
CompilaciÃ³n TAC -> ASM completada.

[3/3] Ejecutando VM...
------------------------------------------
El valor es distinto de 2
------------------------------------------

[PROCESO TERMINADO CON EXITO]
```

Successfully completed (Console)



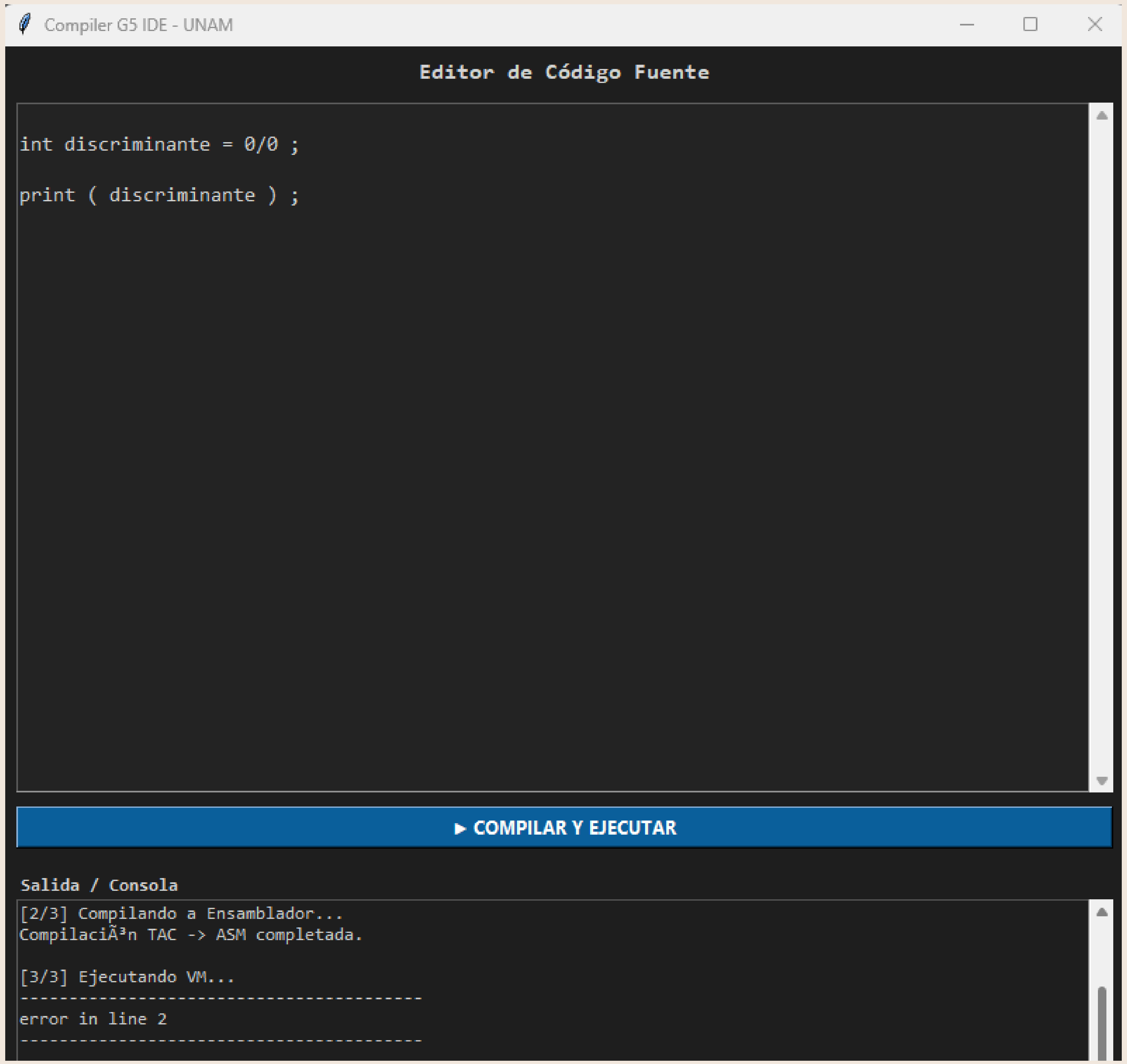
```
PS C:\Users\alons\Documents\Proyecto\unam.fi.compilers.g5.04> ./run.bat examples\temp_gui.txt

[1/3] Lexer - Parser (Analisis)...
--------------------------------
[OK] Sintaxis Correcta. Generando TAC.

[2/3] Compilando a Ensamblador...
Compilaci├│n TAC -> ASM completada.

[3/3] Ejecutando VM...
-----------------------------------------
Result of operation: 0
Result of operation: 1
Result of operation: 2
Result of operation: 3
Result of operation: 4
-----------------------------------------
PS C:\Users\alons\Documents\Proyecto\unam.fi.compilers.g5.04> |
```

Operation failed (Console)



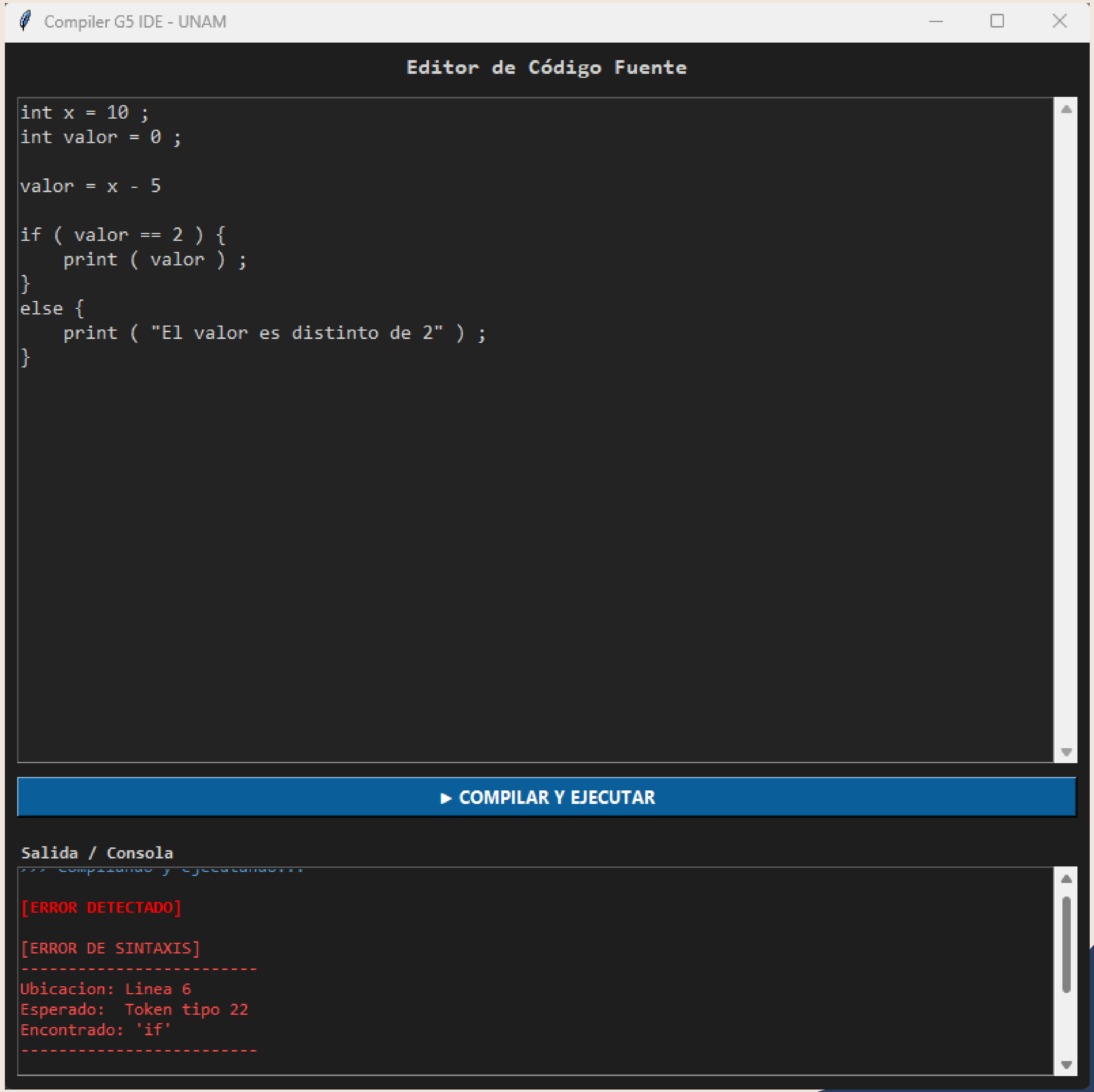
```
PS C:\Users\alons\Documents\Proyecto\unam.fi.compilers.g5.04> ./run.bat examples\temp_gui.txt

[1/3] Lexer - Parser (Analisis)...
---------------------------------


[ERROR DE SINTAXIS]
--------------------------
Ubicacion: Linea 2
Esperado:  Token tipo 22
Encontrado: 'while'
--------------------------


[X] SE ENCONTRO UN ERROR. PROCESO DETENIDO.
Revisa el mensaje de arriba para mas detalles.
```

OPERATION FAILED



```
int discriminante = 0/0 ;

print ( discriminante ) ;
```

```
[2/3] Compilando a Ensamblador...
CompilaciÃ³n TAC -> ASM completada.

[3/3] Ejecutando VM...
----------------------------------------
error in line 2
----------------------------------------
```



```
int x = 10 ;
int valor = 0 ;

valor = x - 5

if ( valor == 2 ) {
    print ( valor ) ;
}
else {
    print ( "El valor es distinto de 2" ) ;
}
```

```
[ERROR DETECTADO]

[ERROR DE SINTAXIS]
-------------------------
Ubicacion: Linea 6
Esperado:  Token tipo 22
Encontrado: 'if'
-------------------------
```

CONCLUSION

This project demonstrates a reliable, efficient, and fully functional compilation pipeline capable of analyzing, validating, and executing high-level code with precision. By integrating lexical, syntactic, and semantic analysis with TAC generation and virtual-machine execution, we deliver a tool that ensures correctness, prevents critical errors early, and guarantees consistent program behavior. The system's modular design makes it scalable, maintainable, and ready to extend toward larger languages or more complex applications. Overall, this solution provides a solid, professional-grade foundation for any organization seeking to build or enhance a custom compiler or automated code-analysis tool.

THANK YOU

FOR YOUR ATTENTION