



Tecnológico de Monterrey

LENGUAJES Y TRADUCTORES

Proyecto Final



Diego Alejandro Garza Gutiérrez
ISD A01039429

Instructora: Dra. Norma Frida Roffe
Semestre Agosto - Diciembre 2020

Introducción	2
Diseño descriptivo del lenguaje	3
Diagramas de sintaxis	4
Léxico y sintaxis	4
Programación del analizador de léxico y sintaxis	5
Tabla de símbolos	5
Generación de código: Expresiones y asignaciones a variables	5
Generación de código: Lenguaje con variables simples	5
Ejecución de asignaciones a variables simples, incluye expresiones aritméticas y booleanas	5
Ejecución de IFs	5
Ejecución de ciclos	5
Proyecto completo	5
Conclusión	5
Referencias	5
Anexo 1	5

Introducción

El objetivo principal de éste proyecto consiste en crear un lenguaje de programación que tenga algunas instrucciones similares los lenguajes como Go, R, Ruby, en su diseño del lenguaje se deberá mostrar de forma clara sus las similitudes. El propósito de mi lenguaje es constituir una herramienta de programación simple, con orientación a cálculos numéricos, útil para un programador principiante.

Los elementos necesarios para llevar acabo el desarrollo de éste lenguaje es el siguiente:

- Encabezado del programa
- Estatutos
 - Asignación de expresiones a variables
 - Lectura de variables
 - Escritura de variables y strings
 - Condicional
 - Dos Ciclos Condicionales (while)
 - Un Ciclo Controlado: (for)
 - Llamadas a módulos sin paso de parámetros
- Módulos
 - Los módulos tendrán un nombre.... pero no parámetros ni variables locales. Pueden localizarse antes o después del programa principal. En un programa puede haber de 0 a n módulos
 - Las llamadas a módulos, deben localizarse en los estatutos:
 - Definiré las palabras reservadas.
 - Los operadores permitidos en las expresiones son:
 - Numeric Operators: () ^ * / + -
 - Relational Operators < <= <> = == >
 - Logical Operators AND OR NOT.
 - Las expresiones deben aceptar cualquier cantidad de operadores, paréntesis anidados y referencias a variables dimensionadas.
 - Usaré la prioridad de operadores
 - Los estatutos comprendidos en los puntos anteriores deben permitir más de un estatuto en su estructura.
 - Permita comentarios en el programa.
- Tipos de Datos
 - Simples:
 - Al menos dos tipos de datos, decidiré si se deben declarar las variables o no.
 - Compuestos:
 - Vectores, Matrices y Cubos (Variables dimensionadas de 1, 2 y 3 dimensiones). Debe haber una declaración para este tipo de variables, para definir el tamaño de las dimensiones.
 - Todas las variables son globales
 - **NOTA:** La declaración de las dimensiones es numérica, pero la referencia es a través de expresiones aritméticas, por ejemplo, debe ser válido algo así como Matriz(i+1,i-1).
- Extra
 - Se podrán obtener puntos extras cuando el lenguaje agregue alguna facilidad nueva (inventada) o añada una aplicación particular.

Diseño descriptivo del lenguaje

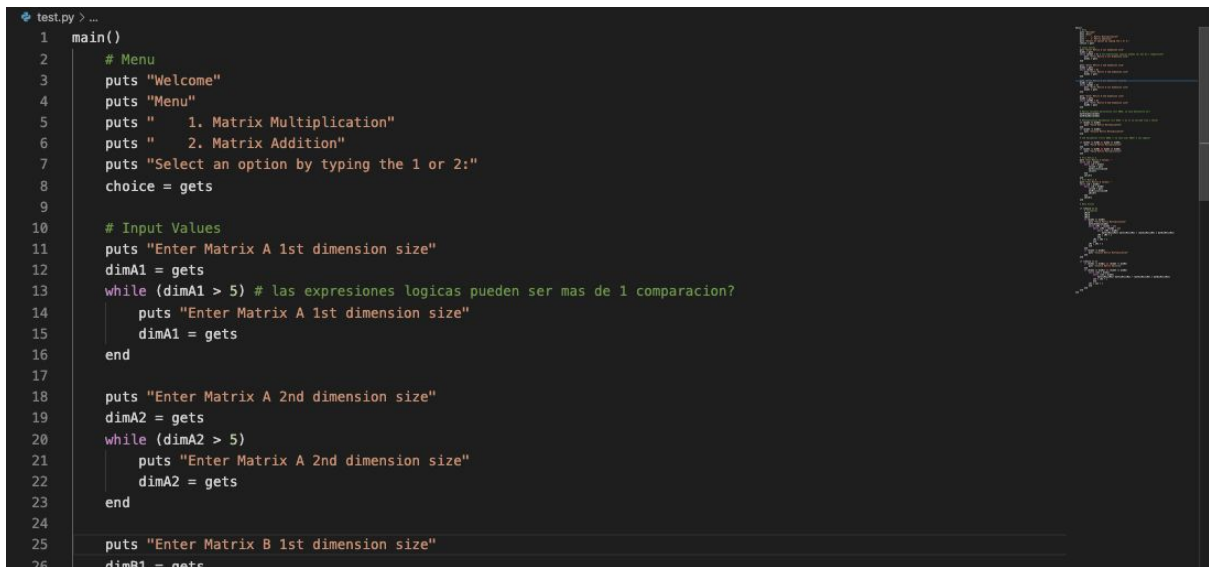
Lenguaje Sunbeam (algunas instrucciones similares a Ruby)

- Encabezado del programa
 - `main() estatutos end`
- Estatutos
 - Escritura de variables y strings
 - Los estatutos comprendidos en los puntos anteriores deben permitir más de un estatuto en su estructura.
 - Asignaciones
 - `variable = expresión aritmética`
 - `variable[x][y]={x1, x2, ... ; x3, x4, ...}`
 - Lectura de variables (**Ruby like**)
 - `puts {variable}`
 - `{variable} = gets`
 - Condicional
 - `if expresión lógica estatutos end` (**Ruby like**)
 - Ciclos Condicionales
 - `while (expresión lógica) estatutos end`
 - `dwhile (expresión lógica) estatutos end`
 - Ciclo Controlado
 - `for (estatutos, expresión lógica, step) estatutos end`
- Módulos
 - Funciones:
 - `function <id>() estatutos end`
 - Los módulos tendrán un nombre.... pero no parámetros ni variables locales. Pueden localizarse antes o después del programa principal. En un programa puede haber de 0 a n módulos
 - Las expresiones deben aceptar cualquier cantidad de operadores, paréntesis anidados y referencias a variables dimensionadas.
 - Palabras reservadas
 - `true, false, and, or ,not, if, then, else, while, dwhile, for, print, main, begin, end, function, until`
 - Los operadores permitidos en las expresiones son (prioridad de operadores):
 - Numeric Operators: `() ^ * / + -`
 - Relational Operators `< <= <> = => >`
 - Logical Operators
 - `and`
 - `or`
 - `not`
 - Comentarios
 - `# Este es un comentario`
- Tipos de Datos
 - Simple:
 - Enteros y flotantes
 - Compuestos:
 - Vectores y Matrices (Variables dimensionadas de 1, 2 y 3 dimensiones)
 - Todas las variables son globales

- NOTA: La declaración de las dimensiones es numérica, pero la referencia es a través de expresiones aritméticas, por ejemplo, debe ser válido algo así como Matriz(i+1,i-1).

Utilizando el lenguaje, diseñaré y mostraré un programa que lea las dimensiones de dos matrices de tamaño máximo de 5x5, y las lea. El programa deberá contar con un menú. El usuario elige la opción de:

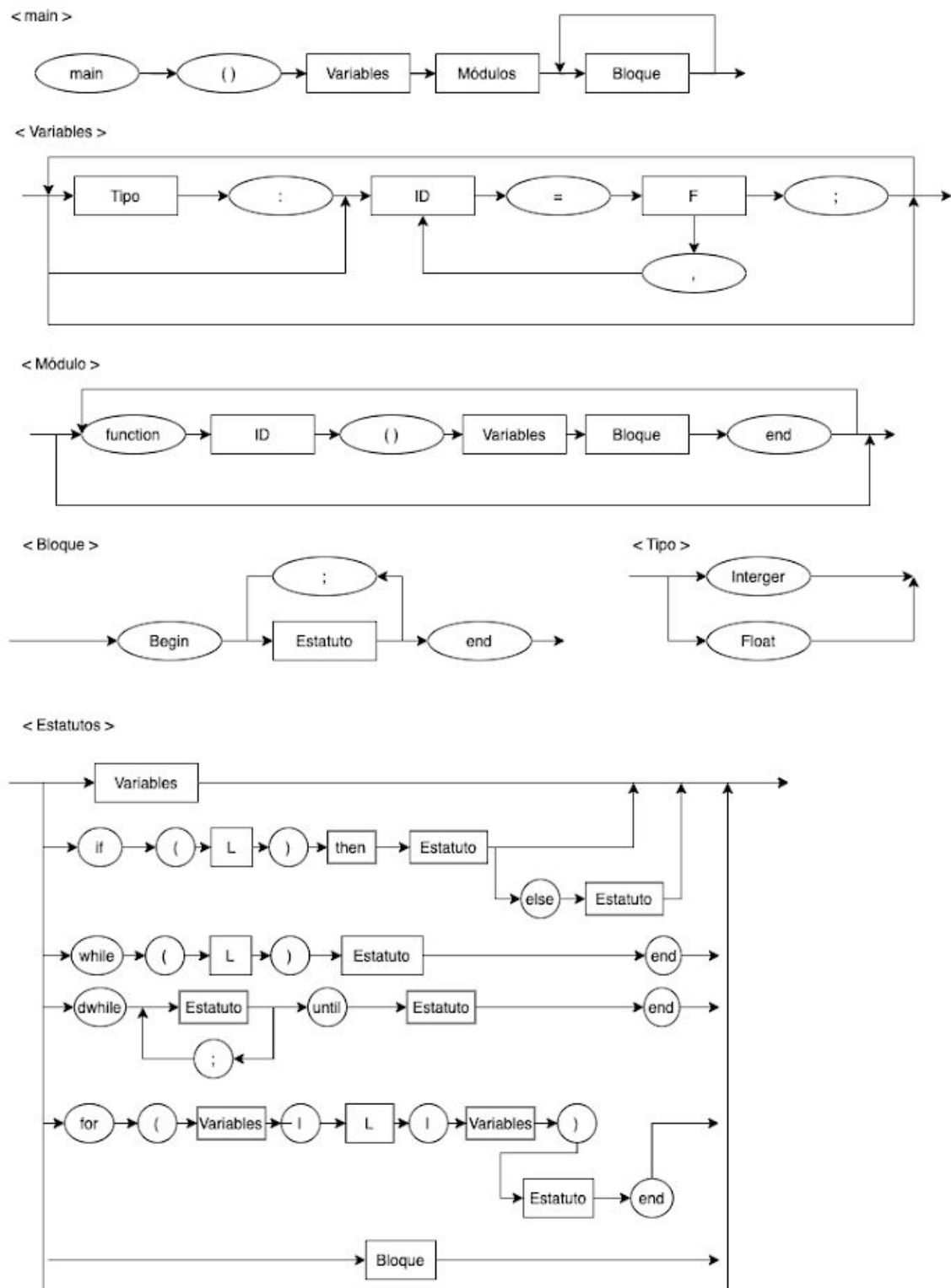
1. Multiplicar las dos matrices y dejar el resultado en una tercera matriz, imprimir resultado. Antes deberá verificar que sea posible realizar la multiplicación.
2. Sumar las dos matrices y dejar el resultado en una tercera matriz, imprimir resultado. Antes deberá verificar que sea posible realizar la suma. Utilice módulos (y llamadas a módulos).

A screenshot of a code editor with a dark theme. The file is named 'test.py'. The code is a Python script with a 'main()' function. It starts with a menu that prints 'Welcome' and 'Menu', then lists two options: '1. Matrix Multiplication' and '2. Matrix Addition'. It prompts the user to 'Select an option by typing the 1 or 2:'. The code then handles input for 'Matrix A' dimensions, with a while loop ensuring the first dimension is not greater than 5. It prompts for the first and second dimensions of Matrix A. The code for Matrix B is partially visible at the bottom.

```
1 main()
2 # Menu
3 puts "Welcome"
4 puts "Menu"
5 puts "  1. Matrix Multiplication"
6 puts "  2. Matrix Addition"
7 puts "Select an option by typing the 1 or 2:"
8 choice = gets
9
10 # Input Values
11 puts "Enter Matrix A 1st dimension size"
12 dimA1 = gets
13 while (dimA1 > 5) # las expresiones logicas pueden ser mas de 1 comparacion?
14     puts "Enter Matrix A 1st dimension size"
15     dimA1 = gets
16 end
17
18 puts "Enter Matrix A 2nd dimension size"
19 dimA2 = gets
20 while (dimA2 > 5)
21     puts "Enter Matrix A 2nd dimension size"
22     dimA2 = gets
23 end
24
25 puts "Enter Matrix B 1st dimension size"
26 dimB1 = gets
```

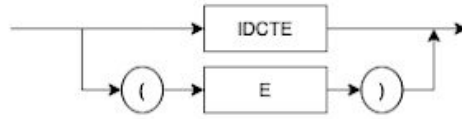
Código completo en documento *"CodigoLenguajeSunbeam.txt"*

Diagramas de sintaxis

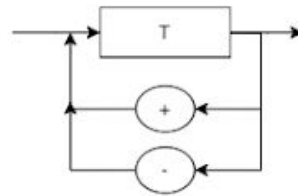


(Expresión Arimética)

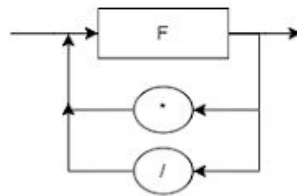
<F>



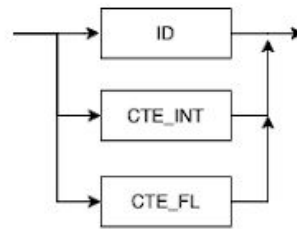
<E>



<T>

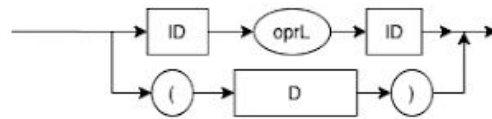


<IDCTE>

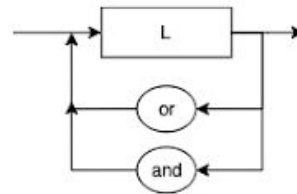


(Expresión Lógica)

<L>



<D>



Léxico y sintaxis

```
S : main

main : MAIN LPAREN RPAREN vars MODULO main1

main1 : BLOQUE
      | BLOQUE main1

vars : decl ID EQUAL vars1 SEMICOLON vars
     |

decl : TIPO COLON
     |

vars1 : F
      | F COMA ID EQUAL vars1

MODULO : FUNCTION ID LPAREN RPAREN vars BLOQUE END MODULO
       |

BLOQUE : BEGIN ESTATUTO BR END
       |
       |'''

BR : SEMICOLON ESTATUTO BR
   |

ESTATUTO : vars
          | IF LPAREN L RPAREN THEN ESTATUTO IF1
          | WHILE LPAREN L RPAREN ESTATUTO END
          | DWHILE ESTATUTO DW1 UNTIL ESTATUTO END
          | FOR LPAREN vars PIPE L PIPE vars RPAREN ESTATUTO END
          | BLOQUE
          |

IF1 : ELSE ESTATUTO
    |

DW1 : SEMICOLON ESTATUTO DW1
    |

F : IDCTE
  | LPAREN E RPAREN

E : T
  | T PLUS E
  | T MINUS E

T : F
  | F TIMES T
  | F DIVIDE T

IDCTE : ID
      | CTE_INT
      | CTE_FL

TIPO : INT
     | FL

L : ID OPRL ID
  | LPAREN D RPAREN

D : L
  | L D1

D1 : OR L
    | AND L
    |

OPRL : GT
     | LT
     | ET
```


Programación del analizador de léxico y sintaxis

En esta sección se trabajó el analizador léxico usando el lenguaje Python. Se desarrolló y se definieron palabras reservadas, tokens, reglas, y la semántica del lenguaje Sunbeam. El código funcionó correctamente, ya que solo desplegará "CORRECTO" cuando todo el sintaxis esta correcto. Es importante recalcar que el código siempre imprime los tokens a como los va reconociendo.

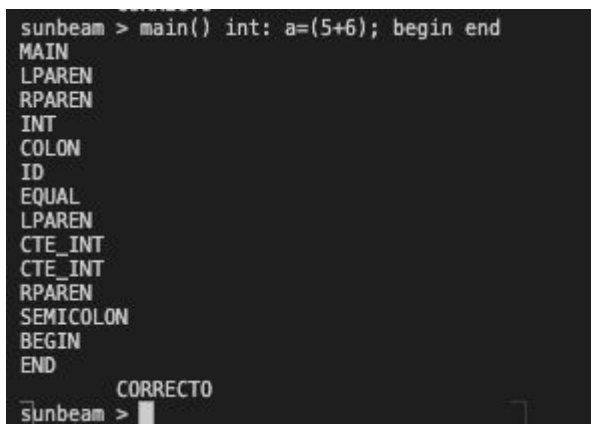
El código se encuentra en el Anexo 2, así como adjunto con el nombre "Sunbeam.py".

Para evidenciar este avance se corrió el programa y se escribió probaron los siguientes programas prueba, los cuales fueron satisfactorios.

Programas:

```
main() int: a=(5+6); begin end
main() int: a=(5),b=(4); function a() a=b; begin a=1; end end
main() int: a=(5),b=(4); function a() a=b; begin if (a<b) then a=12; end end
main() int: a=(5),b=(4); function a() a=b; begin while (a>b) a=4; end end end
main() int: a=(5),b=(4); function a() a=b; begin dwhile a=4; until a=4; end end end
main() int: a=(5),b=(4); fl: c=23.4; function a() a=b; begin for (a=4;|a>b|a=(a+3);) a=5; end end end
```

Salidas:

A screenshot of a terminal window showing the output of the Sunbeam interpreter. The prompt is 'sunbeam >'. The input is 'main() int: a=(5+6); begin end'. The output lists the tokens recognized: MAIN, LPAREN, RPAREN, INT, COLON, ID, EQUAL, LPAREN, CTE_INT, CTE_INT, RPAREN, SEMICOLON, BEGIN, and END. At the bottom, it says 'CORRECTO' and the prompt 'sunbeam >' is shown again with a cursor.

```
sunbeam > main() int: a=(5+6); begin end
MAIN
LPAREN
RPAREN
INT
COLON
ID
EQUAL
LPAREN
CTE_INT
CTE_INT
RPAREN
SEMICOLON
BEGIN
END
CORRECTO
sunbeam > 
```

```

sunbeam > main() int: a=(5),b=(4); function a() a=b; begin a=1; end end
MAIN
LPAREN
RPAREN
INT
COLON
ID
EQUAL
LPAREN
CTE_INT
RPAREN
ID
EQUAL
LPAREN
CTE_INT
RPAREN
SEMICOLON
FUNCTION
ID
LPAREN
RPAREN
ID
EQUAL
ID
SEMICOLON
BEGIN
ID
EQUAL
CTE_INT
SEMICOLON
END
END
CORRECTO
sunbeam > █

```

```

sunbeam > main() int: a=(5),b=(4); function a() a=b; begin if (a<b) then a=12; end end
MAIN
LPAREN
RPAREN
INT
COLON
ID
EQUAL
LPAREN
CTE_INT
RPAREN
ID
EQUAL
LPAREN
CTE_INT
RPAREN
SEMICOLON
FUNCTION
ID
LPAREN
RPAREN
ID
EQUAL
ID
SEMICOLON
BEGIN
IF
LPAREN
ID
LT
ID
RPAREN
THEN
ID
EQUAL
CTE_INT
SEMICOLON
END
END
CORRECTO
sunbeam > █

```

```
sunbeam > main() int: a=(5),b=(4); function a() a=b; begin while (a>b) a=4; end end end
MAIN
LPAREN
RPAREN
INT
COLON
ID
EQUAL
LPAREN
CTE_INT
RPAREN
ID
EQUAL
LPAREN
CTE_INT
RPAREN
SEMICOLON
FUNCTION
ID
LPAREN
RPAREN
ID
EQUAL
ID
SEMICOLON
BEGIN
WHILE
LPAREN
ID
GT
ID
RPAREN
ID
EQUAL
CTE_INT
SEMICOLON
END
END
END
CORRECTO
sunbeam > █
```

```

sunbeam > main() int: a=(5),b=(4); function a() a=b; begin dwhile a=4; until a=4; end end end
MAIN
LPAREN
RPAREN
INT
COLON
ID
EQUAL
LPAREN
CTE_INT
RPAREN
ID
EQUAL
LPAREN
CTE_INT
RPAREN
SEMICOLON
FUNCTION
ID
LPAREN
RPAREN
ID
EQUAL
ID
SEMICOLON
BEGIN
DWHILE
ID
EQUAL
CTE_INT
SEMICOLON
UNTIL
ID
EQUAL
CTE_INT
SEMICOLON
END
END
END
CORRECTO
sunbeam >

```

Ln 220, Col 8 Tab Size

```

CORRECTO
sunbeam > main() int: a=(5),b=(4); fl: c=23.4; function a() a=b; begin for (a=4;|a>b|a=(a+3);) a=5; end end end
MAIN
LPAREN
RPAREN
INT
COLON
ID
EQUAL
LPAREN
CTE_INT
RPAREN
ID
EQUAL
LPAREN
CTE_INT
RPAREN
SEMICOLON
FL
COLON
ID
EQUAL
CTE_FL
SEMICOLON
FUNCTION
ID
LPAREN
RPAREN
ID
EQUAL
ID
SEMICOLON
BEGIN
FOR
LPAREN
ID
EQUAL
CTE_INT
SEMICOLON
PIPE
ID
GT
ID
PIPE
ID
EQUAL
LPAREN
ID
CTE_INT
RPAREN
SEMICOLON
RPAREN
ID
EQUAL
CTE_INT
SEMICOLON
END
END
END
CORRECTO
sunbeam >

```

Tabla de símbolos

Generación de código: Expresiones y asignaciones a variables

Generación de código: Lenguaje con variables simples

Ejecución de asignaciones a variables simples, incluye expresiones aritméticas y booleanas

Ejecución de IFs

Ejecución de ciclos

Proyecto completo

Conclusión

Referencias

Anexo 1

```
main()
# Menu
puts "Welcome"
puts "Menu"
puts "    1. Matrix Multiplication"
puts "    2. Matrix Addition"
puts "Select an option by typing the 1 or 2:"
choice = gets

# Input Values
puts "Enter Matrix A 1st dimension size"
dimA1 = gets
while (dimA1 > 5) # las expresiones logicas pueden ser mas de 1 comparacion?
    puts "Enter Matrix A 1st dimension size"
    dimA1 = gets
end
```

```

puts "Enter Matrix A 2nd dimension size"
dimA2 = gets
while (dimA2 > 5)
    puts "Enter Matrix A 2nd dimension size"
    dimA2 = gets
end

puts "Enter Matrix B 1st dimension size"
dimB1 = gets
while (dimB1 > 5)
    puts "Enter Matrix B 1st dimension size"
    dimA1 = gets
end

puts "Enter Matrix B 2nd dimension size"
dimB2 = gets
while (dimB2 > 5)
    puts "Enter Matrix B 2nd dimension size"
    dimB2 = gets
end

# Matrix variable declaration **** DUDA, se vale declararla así?
matA[dimA1][dimA2]
matB[dimB1][dimB2]

# Multiplication Validation **** DUDA -> el if se escribe true y false?
if (dimA2 == dimB1)
    puts "Valid Matrix Multiplication"
end
if (dimA2 != dimB1)
    puts "Invalid Matrix Multiplication"
end

# Sum Validation ***** DUDA -> se vale usar ANDs? o los separo?

if (dimA1 == dimB1 && dimA2 == dimB2)
    puts "Valid Matrix Multiplication"
end
if (dimA1 != dimB1 && dimA2 == dimB2)
    puts "Valid Matrix Multiplication"
end

# Fill Matrix A
puts "Fill Matrix A values: "
while (iA < dimA1)
    while (jA < dimA2)
        valueA = gets
        matA[i][j]=valueA
        jA=jA+1
    end
    iA=iA+1
end

# Fill Matrix B
puts "Fill Matrix B values: "
while (iB < dimB1)
    while (jB < dimB2)
        valueB = gets
        matB[i][j]=valueB
        jB=jB+1
    end
    iB=iB+1
end
end

```

```

# Menu Action

if (choice == 1)
    # validation
    iM1=0
    iM2=0
    iM3=0
    if (dimA2 == dimB1)
        puts "Valid Matrix Multiplication"
        matC[dimA1][dimB2]
        while (iM1 < (dimA1 - 1))
            while (iM2 < (dimB2 - 1))
                while (iM3 < dimB1)
                    matC[iM1][iM2]= matC[iM1][iM2] + (matA[iM1][iM3] * matB[iM3][iM2])
                    iM3 = iM3 + 1
                end
                iM2 = iM2 + 1
            end
            iM1 = iM1 + 1
        end
    end
    if (dimA2 != dimB1)
        puts "Invalid Matrix Multiplication"
    end
end

if (choice == 2)
    if (dimA1 != dimB1) && (dimA2 != dimB2)
        puts "Invalid Matrix Adittion"
    end
    if (dimA1 == dimB1) && (dimA2 == dimB2)
        while (iA1 < dimA1)
            while (iA2 < dimA2)
                matC[iM1][iM2]= matC[iM1][iM2] + (matA[iM1][iM2] * matB[iM1][iM2])
                iA2 = iA2 + 1
            end
            iA1 = iA1 + 1
        end
    end
end
end
end

```

Anexo 2

```

#Proyecto Lenguajes y traductores
import ply.lex as lex
import ply.yacc as yacc
import sys

reserved = {
    'if' : 'IF',
    'then' : 'THEN',
    'else' : 'ELSE',
    'while' : 'WHILE',
    'dwhile' : 'DWHILE',
    'for' : 'FOR',
    'until' : 'UNTIL',
    'function' : 'FUNCTION',
    'int' : 'INT',
    'fl' : 'FL',
    'and' : 'AND',
    'or' : 'OR'
}

tokens = [
    'MAIN',
    'BEGIN',
    'END',

```



```

        'ID',
        'NUMBER',
        'PLUS',
        'MINUS',
        'TIMES',
        'DIVIDE',
        'LPAREN',
        'RPAREN',
        'COLON',
        'SEMICOLON',
        'EQUAL',
        'PIPE',
        'GT',
        'LT',
        'ET',
        'CTE_FL',
        'CTE_INT',
        'CHAR',
        'COMA'

] + list(reserved.values())

t_ignore = r' '
t_ignore = r'\t'
t_ignore_COMMENT = r'\#.*'

t_PLUS = r'\+'
t_MINUS = r'\-'
t_TIMES = r'\*'
t_DIVIDE = r'\/'
t_COMA = r','

def t_MAIN(t):
    r'main'
    t.type = 'MAIN'
    print(t.type)
    return t

def t_BEGIN(t):
    r'begin'
    t.type = 'BEGIN'
    print("BEGIN")
    return t

def t_END(t):
    r'end'
    t.type = 'END'
    print("END")
    return t

def t_LPAREN(t):
    r'\('
    t.type = 'LPAREN'
    print("LPAREN")
    return t

def t_RPAREN(t):
    r'\)'
    t.type = 'RPAREN'
    print("RPAREN")
    return t

def t_PIPE(t):
    r'\|'
    t.type = 'PIPE'
    print("PIPE")
    return t

def t_COLON(t):
    r'\:'
    t.type = 'COLON'
    print("COLON")
    return t

def t_SEMICOLON(t):
    r'\;'
    t.type = 'SEMICOLON'
    print("SEMICOLON")
    return t

def t_EQUAL(t):
    r'\='
    t.type = 'EQUAL'
    print("EQUAL")
    return t

def t_GT(t):
    r'\>'
    t.type = 'GT'
    print("GT")
    return t

def t_LT(t):
    r'\<'
    t.type = 'LT'
    print("LT")
    return t

def t_ET(t):
    r'\=='
    t.type = 'ET'
    print("ET")
    return t

def t_ID(t):
    r'[a-zA-Z][a-zA-Z_0-9]*'
    t.type = reserved.get(t.value, 'ID')
    print(t.type)
    return t

def t_CTE_FL(t):
    r'\d+\.\d+'
    t.value = float(t.value)
    print(t.type)
    return t

def t_CTE_INT(t):
    r'\d+'
    t.value = int(t.value)

```

```

        print(t.type)
        return t

def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)

def t_error(t):
    print("Illegal characters!")
    t.lexer.skip(1)

lexer = lex.lex(optimize=1)

def p_S(p):
    """
    S : main
    """
    print("\tCORRECTO")

def p_main(p):
    """
    main : MAIN LPAREN RPAREN vars MODULO main1
    """
    pass

def p_main1(p):
    """
    main1 : BLOQUE
          | BLOQUE main1
    """
    pass

def p_vars(p):
    """
    vars : decl ID EQUAL vars1 SEMICOLON vars
          |
    """
    pass

def p_decl(p):
    """
    decl : TIPO COLON
          |
    """
    pass

def p_vars1(p):
    """
    vars1 : F
          | F COMA ID EQUAL vars1
    """
    pass

def p_MODULO(p):
    """
    MODULO : FUNCTION ID LPAREN RPAREN vars BLOQUE END MODULO
    """
    pass

def p_BLOQUE(p):
    """
    BLOQUE : BEGIN ESTATUTO BR END
    """
    pass

def p_BR(p):
    """
    BR : SEMICOLON ESTATUTO BR
    """
    pass

def p_ESTATUTO(p):
    """
    ESTATUTO : vars
              | IF LPAREN L RPAREN THEN ESTATUTO IF1
              | WHILE LPAREN L RPAREN ESTATUTO END
              | DWHILE ESTATUTO DW1 UNTIL ESTATUTO END
              | FOR LPAREN vars PIPE L PIPE vars RPAREN ESTATUTO END
              | BLOQUE
    """
    pass

def p_IF1(p):
    """
    IF1 : ELSE ESTATUTO
    """
    pass

def p_DW1(p):
    """
    DW1 : SEMICOLON ESTATUTO DW1
    """
    pass

def p_X(p):
    """
    X : MAIN
      | BEGIN
      | END
    """
    pass

def p_F(p):
    """
    F : IDCTE
      | LPAREN E RPAREN
    """
    pass

def p_E(p):
    """
    E : T
      | T PLUS E
      | T MINUS E
    """
    pass

def p_T(p):
    """
    T : F
      | F TIMES T
      | F DIVIDE T
    """
    pass

def p_IDCTE(p):
    """
    IDCTE : ID
           | CTE_INT
           | CTE_FL
    """
    pass

def p_TIPO(p):

```

```

        '''
        TIPO : INT
              | FL
        '''

def p_L(p):
    '''
    L : ID OPRL ID
      | LPAREN D RPAREN
    '''

def p_D(p):
    '''
    D : L
      | L D1
    '''

def p_D1(p):
    '''
    D1 : OR L
        | AND L
    '''

def p_OPRL(p):
    '''
    OPRL : GT
          | LT
          | ET
    '''

def p_error(p):
    print(f"Syntax error at {p.value!r}")

parser = yacc.yacc()

while True:
    try:
        s = input('sunbeam > ')
    except EOFError:
        break
    parser.parse(s)

```