Tecnológico de Monterrey

Proyecto Final de Compiladores

Maestra Elda Quiroga

Lenguaje SLOW

Diego Alejandro Villarreal López
A01282555

Rubén Alejandro López Charles
A00819291

06 de junio del 2022

ÍNDICE

Descripción y documentación del proyecto

DESCRIPCIÓN DEL PROYECTO

- Propósito
    - El propósito del desarrollo del lenguaje fue el aprendizaje y refuerzo del manejo en Python y la herramienta de parseo PLY (Python Lex-Yacc). Asimismo reforzar lo aprendido en la materia de *Diseño de Compiladores*.

- Análisis de Requerimientos y descripción de los principales test cases
    - El lenguaje deberá soportar los estatutos básicos de un lenguaje de programación, estos siendo:
        - Declaración de variables
        - Declaración de funciones
        - Estatutos de asignación
        - Estatutos condicionales
        - Estatutos de repetición
        - Expresiones aritméticas
        - Expresiones lógicas
        - Lectura (equivalente a cin)
        - Escritura (equivalente a cout)
        - Retorno de valores

- Descripción del proceso general seguido para el desarrollo del proyecto.
    - Se llevaron a cabo reuniones diarias utilizando programación en pareja (o *peer programming* por su nombre en inglés).

      -20 de Mayo del 2022-
      Participantes: Diego Villarreal y Rubén López
      Se hizo debugging extensivo y corrección de errores. La sintaxis general de un programa SIN variables ni funciones ya funciona.

      -25 de Mayo del 2022-
      Participantes: Diego Villarreal y Rubén López
      Se realizó el cubo semántico, tabla de variables y la tabla de funciones.

      -01 de Junio del 2022-
      Participantes: Diego Villarreal y Rubén López
      Se agregó semántica de condicionales y ciclos condicionales.

      - 04 de Junio del 2022-
      Participantes: Participantes: Diego Villarreal y Rubén López

Se realizó un avance de la memoria virtual y se corrigieron errores en las funciones.
-05 de Junio del 2022-
Se hicieron bugfixes en general de las operaciones aritméticas.

-06 de Junio del 2022-
Se hicieron bugfixes de la generación de cuadruplos en los estatutos condicionales y de repetición.

Reflexiones generales

Rubén	Diego
Aprendí bastante sobre las gramáticas y sobre las consideraciones sintácticas, siempre me había disgustado el lenguaje tipo ensamblador, y cuando comenzamos a ver cuadruplos tuve miedo al instante por ser similar, pero por alguna razón me hizo bastante sentido cuando lo vimos y hasta le agarré gusto a la materia en cuanto a sus contenidos.	Este proyecto definitivamente ha sido de los proyectos más difíciles que he hecho en la vida, al principio no me llamaba mucho la atención y no comprendía casi nada de cómo funcionaban las cosas, pero al final lo disfrute mucho ya que entre más código generas, más le entiendes y más te diviertes al hacerlo, pero nos faltó tiempo =(.

Firmas de los participantes

DESCRIPCIÓN DEL LENGUAJE

- Nombre del lenguaje
    - Slow
- Descripción general
    - El lenguaje slow es un lenguaje de programación de alto nivel fuertemente tipado que además es estructural.

- Listado de los errores que pueden ocurrir
    - Si un operador no se encuentra en el punto correcto no se hará append a la pila de operadores y generará error.

- Si dos variables no son admitidas por el cubo semántico generará un error.
- Si nos salimos de la sintaxis generará un error
- Si mandamos un parámetro equivocado a una función de memoria moverá los cuádruplos o generara error
- Si movemos un parámetro equivocado a una función de Tvars generará error
- Si le asignamos a una variable que no existe generará error.
- Si ponemos un paréntesis sin cerrar generará error.

DESCRIPCIÓN DEL COMPILADOR

- Equipo de computo, lenguaje y utilería especiales usadas en el desarrollo del proyecto
  - El desarrollo se llevó a cabo en máquinas de cómputo Windows y MacOS. Se utilizó el lenguaje de programación Python en su versión 3.0 y la herramienta de parseo PLY (Python Lex-Yacc)

- Descripción del análisis léxico
  - Patrones de construcción
    - INT : `r'\d+'`
    - FLOAT : `r'\d+\.\d+'`
    - STRING : `r"(\"([^\\\"]|\\.)+\")|(\'([^\\\']|\\.)+\')"`
    - CHAR : `r"'[a-zA-Z]'"`
    - NEW LINE : `r'\n+'`
    - ID : `r'[a-zA-Z_][a-zA-Z_0-9]*'`

  - Enumeración de los “tokens” y su código asociado

    TOKENS
    - PLUS_OP +
    - MINUS_OP -
    - MULT_OP *
    - DIV_OP /
    - ASSIGN =
    - EQUAL_LOG ==
    - LT_LOG <
    - LTE_LOG <=
    - GT_LOG >
    - GTE_LOG >=
    - NE_LOG <>
    - OR_LOG |
    - AND_LOG &
    - LPAREN (
    - RPAREN)
    - LBRACK [

- RBRACK]
- LCURLY {
- RCURLY }
- COMMA ,
- SEMIC ;
- COLON :
- IGNORE

PALABRAS RESERVADAS

- program : PROGRAM
- vars : VARS
- void : VOID
- main : MAIN
- while : WHILE
- write : WRITE
- read : READ
- return : RETURN
- if : IF
- then : THEN
- else : ELSE
- vars : VARS
- function : FUNCTION
- while : WHILE
- do : DO *no utilizado*
- for : FOR *no utilizado*
- to : TO *no utilizado*
- point : POINT *no utilizado*
- line : LINE *no utilizado*
- arc : ARC *no utilizado*
- penup : PENUP *no utilizado*
- pendown : PENDOWN *no utilizado*
- circle : CIRCLE *no utilizado*
- size : SIZE *no utilizado*

- Descripción del análisis sintáctico
  - Gramática formal empleada

```
# INICIO
def p_program(p):
    'program : PROGRAM ID SEMIC dec_var_gob def_funciones
main'

# VARIABLES Y TIPOS
def p_dec_var_gob(p):
```

```
    '''
    dec_var_gob : VARS tipos COLON lista_ids dec_var_aux
                | empty
    '''

def p_dec_var_aux(p):
    '''
    dec_var_aux : tipos COLON lista_ids SEMIC dec_var_aux
                | empty
    '''

def p_lista_ids(p):
    '''
    lista_ids : ID lista_aux lista_aux_b
    '''

def p_lista_aux(p):
    '''
    lista_aux : LBRACK INT_CTE RBRACK
              | empty
    '''

def p_lista_aux_b(p):
    '''
    lista_aux_b : COMMA lista_ids
                | empty
    '''

def p_tipos(p):
    '''
    tipos   : INT_TYPE
            | FLOAT_TYPE
            | CHAR_TYPE
    '''

# DEFINICION DE FUNCIONES
def p_def_funciones(p):
    '''
    def_funciones : FUNCTION tipos_func ID LPAREN parametros
RPAREN SEMIC dec_var_loc bloque
                  | empty
    '''
```

```
def p_tipos_func(p):
    '''
    tipos_func : INT_TYPE
               | VOID
    '''

def p_dec_var_loc(p):
    '''
    dec_var_loc : VARS tipos COLON ID dec_var_loc_aux SEMIC
var_loc_rec
                | empty
    '''

def p_dec_var_loc_aux(p):
    '''
    dec_var_loc_aux : COMMA ID dec_var_loc_aux
                    | empty
    '''

def p_var_loc_rec(p):
    '''
    var_loc_rec : tipos COLON ID dec_var_loc_aux SEMIC
var_loc_rec
                | empty
    '''

def p_parametros(p):
    '''
    parametros : tipos COLON ID  param_aux
    '''

def p_param_aux(p):
    '''
    param_aux : COMMA parametros
              | empty
    '''

# ESTRUCTURA Y ESTATUTOS

def p_main(p):
    '''
    main : MAIN LPAREN RPAREN  bloque
    '''
```

```
def p_bloque(p):
    '''
    bloque : LCURLY estatutos estatu_rec RCURLY
    '''

def p_estatu_rec(p):
    '''
    estatu_rec : estatutos estatu_rec
               | empty
    '''

def p_estatutos(p):
    '''
    estatutos : asignacion
              | declaracion
              | llamada_func
              | llamada_void
              | retorno
              | lectura
              | escritura
              | decision
              | loop_cond
              | loop_no_cond
              | expresiones
              | empty
    '''

def p_asignacion(p):
    '''
    asignacion : ID ASSIGN expresiones SEMIC
    '''

def p_declaracion(p):
    '''
    declaracion : tipos ID declaracion_aux SEMIC
    '''

def p_declaracion_aux(p):
    '''
    declaracion_aux : ASSIGN expresiones
                    | empty
```

```
    '''

def p_llamada_func(p):
    '''
    llamada_func : ID LPAREN  llamada_func_aux RPAREN SEMIC
    '''

def p_llamada_func_aux(p):
    '''
    llamada_func_aux : expresiones llama_func_auxb
                     | empty
    '''

def p_llama_func_auxb(p):
    '''
    llama_func_auxb : COMMA llamada_func_aux
                    | empty
    '''

def p_llamada_void(p):
    '''
    llamada_void : ID LPAREN llamada_void_aux RPAREN  SEMIC
    '''

def p_llamada_void_aux(p):
    '''
    llamada_void_aux : expresiones llama_void_auxb
                     | empty
    '''

def p_llama_void_auxb(p):
    '''
    llama_void_auxb : COMMA llamada_func_aux
                    | empty
    '''

def p_retorno(p):
    '''
    retorno : RETURN LPAREN expresiones RPAREN SEMIC
    '''

def p_lectura(p):
    '''
```

```
    lectura : READ LPAREN ID lec_aux RPAREN SEMIC
    '''

def p_lec_aux(p):
    '''
    lec_aux : COMMA ID lec_aux
            | empty
    '''

def p_escritura(p):
    '''
    escritura : WRITE LPAREN RPAREN SEMIC
    '''

def p_decision(p):
    '''
    decision : IF LPAREN expresiones RPAREN THEN bloque else
    '''

def p_else(p):
    '''
    else : ELSE bloque
        | empty
    '''

def p_loop_cond(p):
    '''
    loop_cond : WHILE LPAREN expresiones RPAREN DO bloque
    '''

def p_loop_no_cond(p):
    '''
    loop_no_cond : FOR ID nocond_aux ASSIGN expresiones TO
expresiones DO bloque
    '''

def p_nocond_aux(p):
    '''
    nocond_aux : LBRACK expresiones RBRACK
               | empty
    '''

# EXPRESIONES
```

```
def p_expresiones(p):
    '''
    expresiones : t_exp or_check
    '''

def p_or_check(p):
    '''
    or_check : OR_LOG expresiones
             | empty
    '''

def p_t_exp(p):
    '''
    t_exp : g_exp and_check
    '''

def p_and_check(p):
    '''
    and_check : AND_LOG t_exp
            | empty
    '''

def p_g_exp(p):
    '''
    g_exp : m_exp op_check
    '''

def p_op_check(p):
    '''
    op_check : empty
             | comparacion m_exp
    '''

def p_comparacion(p):
    '''
    comparacion : GT_LOG
                | LT_LOG
                | EQUAL_LOG
                | NE_LOG
    '''

def p_m_exp(p):
```

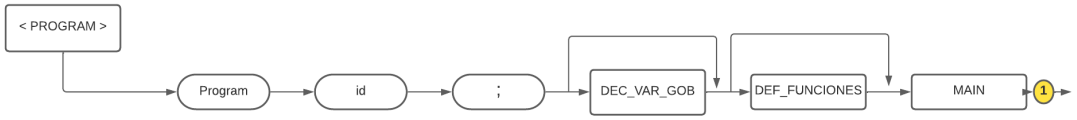
```
    '''
    m_exp : termino m_rec
    '''

def p_m_rec(p):
    '''
    m_rec : PLUS_OP m_exp
          | MINUS_OP m_exp
          | empty
    '''

def p_termino(p):
    '''
    termino :    factor term_rec
    '''

def p_term_rec(p):
    '''
    term_rec : MULT_OP  termino
             | DIV_OP termino
             | empty
    '''

def p_factor(p):
    '''
    factor : LPAREN expresiones RPAREN
           | cte
           | ID
           | llamada_func
    '''
def p_cte(p):
    '''
    cte     : INT_CTE
           | FLOAT_CTE
    '''
```

- Descripción de generación de código intermedio y análisis semántico
    - Código de operación y direcciones virtuales asociadas

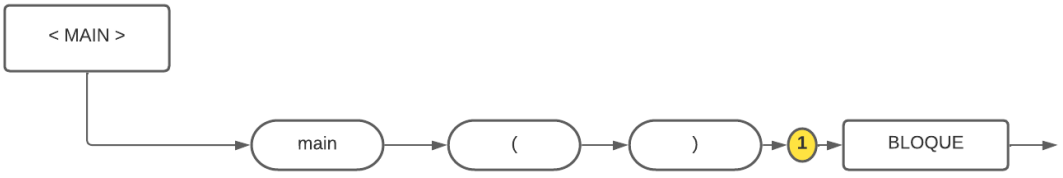
Contexto de la variable o constante	Tipo de variable	Segmento de memoria

Variables Globales	Enteros	0-499
	Flotantes	500-999
	Strings	1000-1499
Variables Locales	Enteros	1500-1999
	Flotantes	2000-2499
	Strings	2500-2999
Variables Temporales	Enteros	3000-3499
	Flotantes	3500-3999
	Strings	4000-4499
	Booleanos	4500-4999
Constantes	Enteros	5000-5499
	Flotantes	5500-5999
	Strings	6000-6499

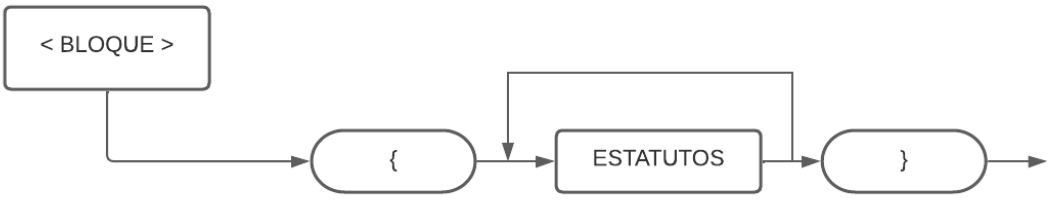
- Diagramas de Sintaxis con las acciones correspondientes marcadas sobre ellos

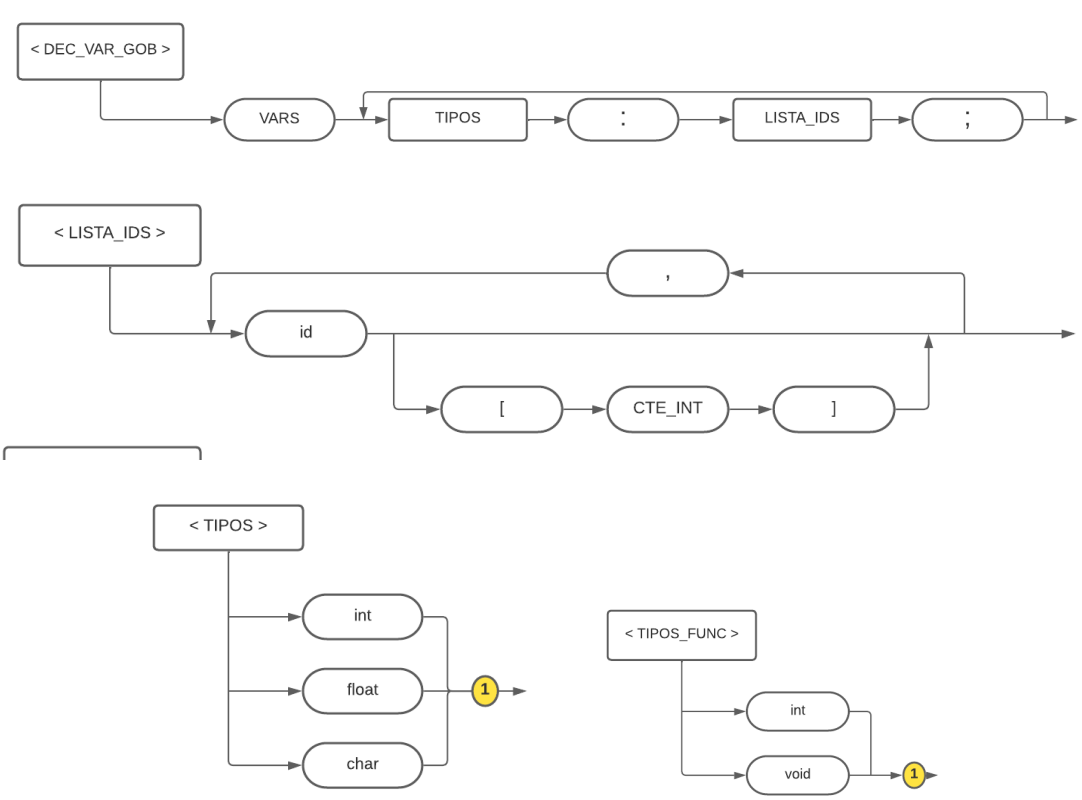


1. Print para pruebas y resultado final

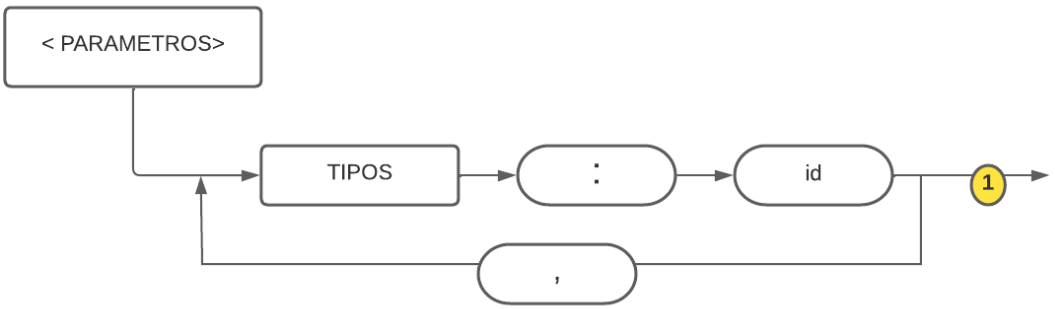


1. Hace el cambio de contexto en la variable global

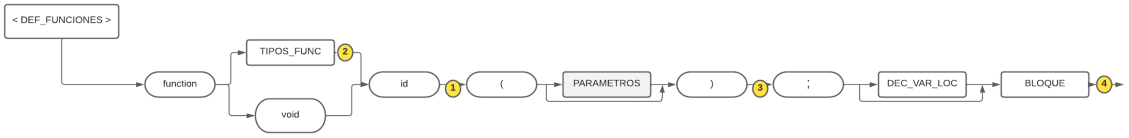




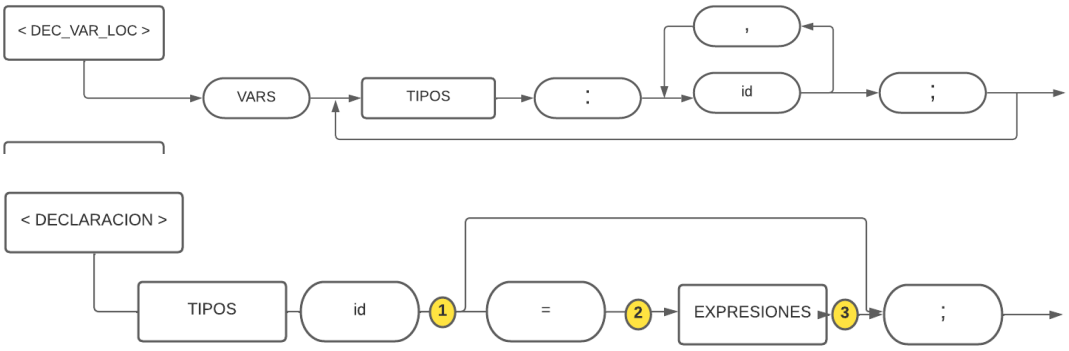
1. En ambos diagramas se hace el cambio en la variable global que maneja el contexto.



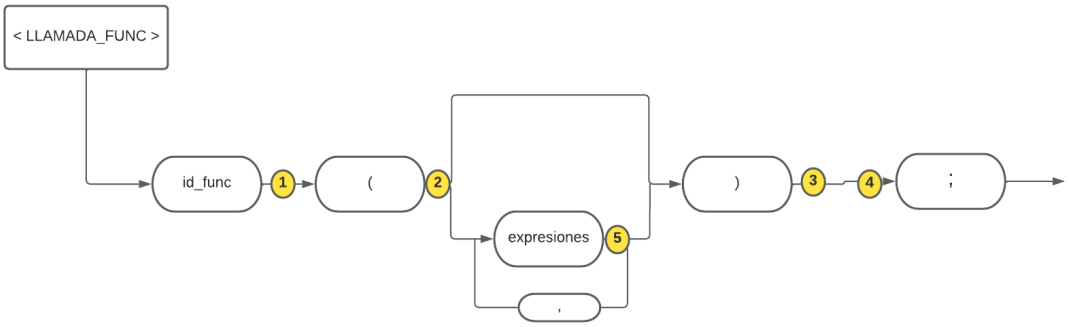
1. Se agregan las variables a la tabla de variables.



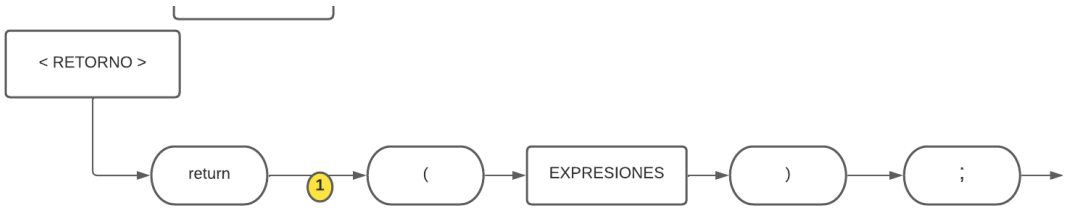
1. Se agrega la función al directorio de funciones.
2. Se actualiza el tipo de la función en la variable global.
3. Se agregan los parámetros a las variables globales.
4. Elimina memoria y genera cuádruplo ENDFUNC



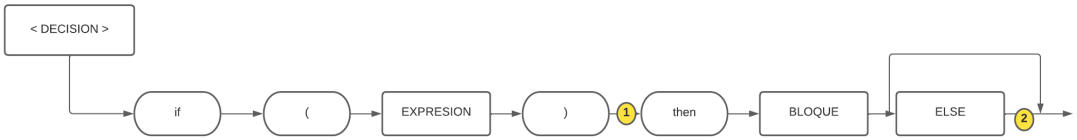
1. Se agrega la variable a la tabla
2. Se agrega el operador a la pila de operadores
3. Se generan los cuádruplos con las operaciones



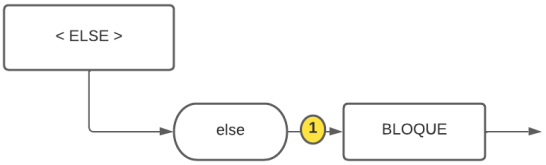
1. Hacemos el ERA para hacer memoria.
2. Append de “(“
3. Parámetros con coma en estilo de cuádruplo PARAM
4. pOperad.pop()
5. Gosub hacia la función



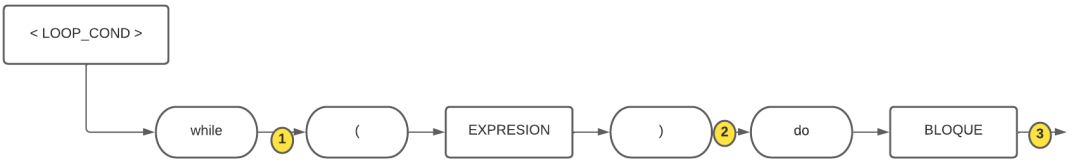
1. Se genera el cuádruplo del return.



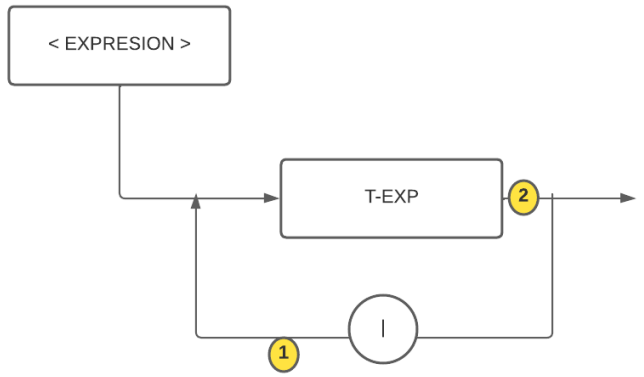
1. Se genera el cuádruplo con el gotoF.
2. Se genera el cuádruplo con el goto.

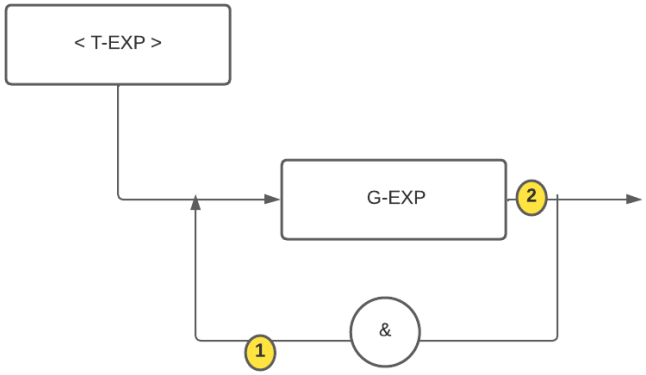


1. Se rellena el cuádruplo generado en el punto 1 con el IP respectivo.

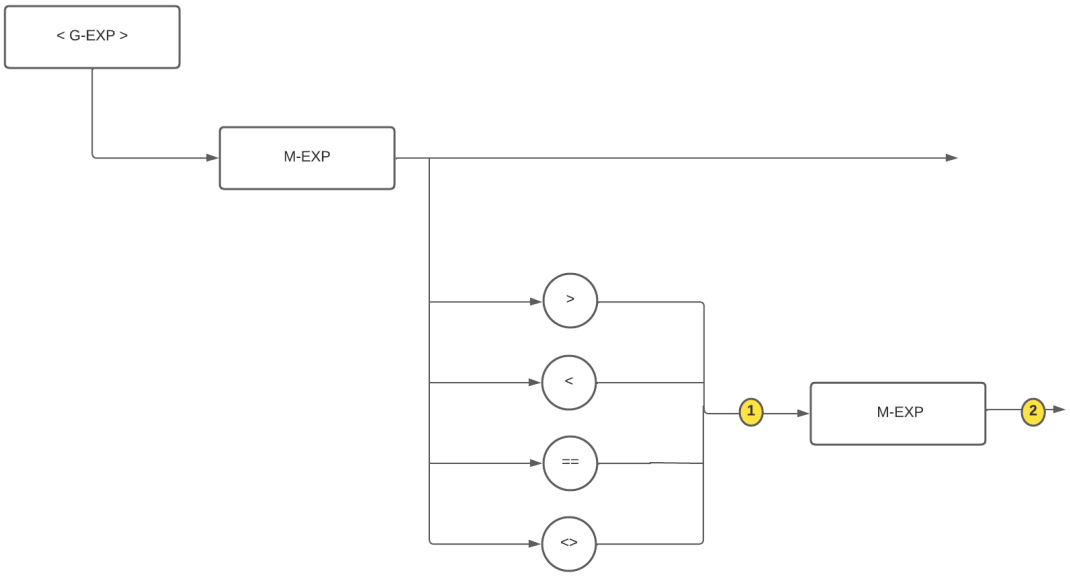


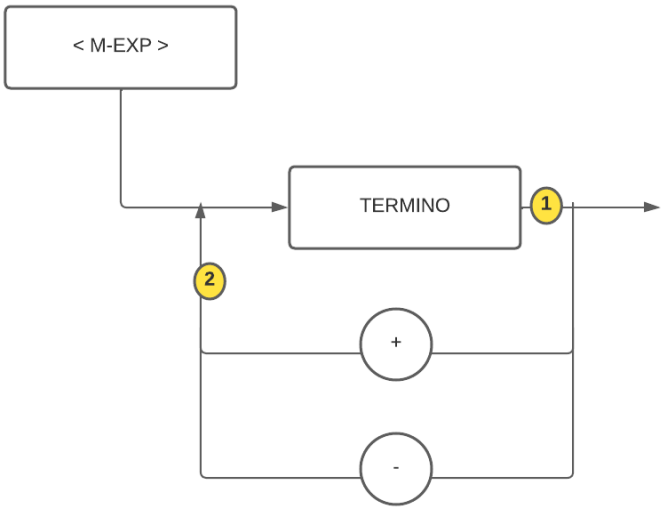
1. Se mete a la pila de saltos.
2. Se genera el cuádruplo con el gotoF .
3. Se genera el cuádruplo del goto.



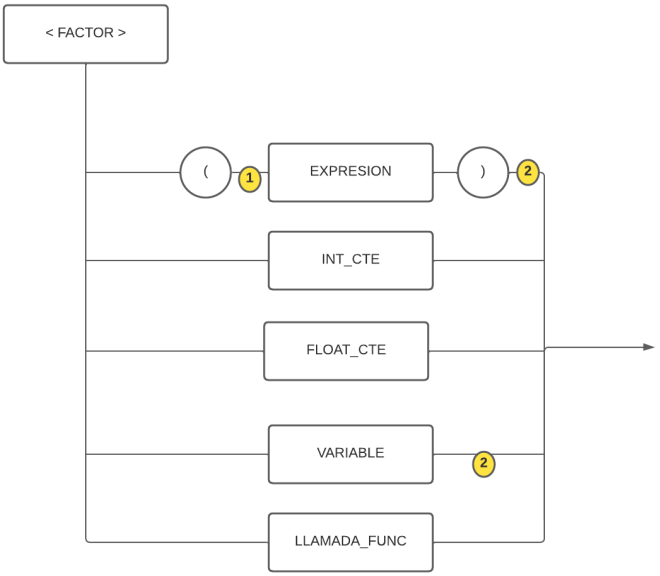


1. hacemos doble pop de tipo, de memoria y de operandos, checamos cubo semántico y generamos cuádruplo
2. Si es un && o un || hacemos append a pila de operadores

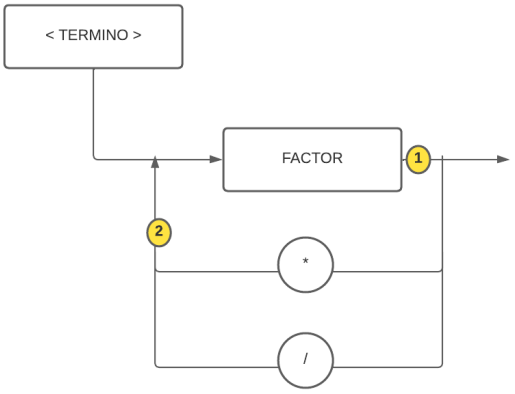




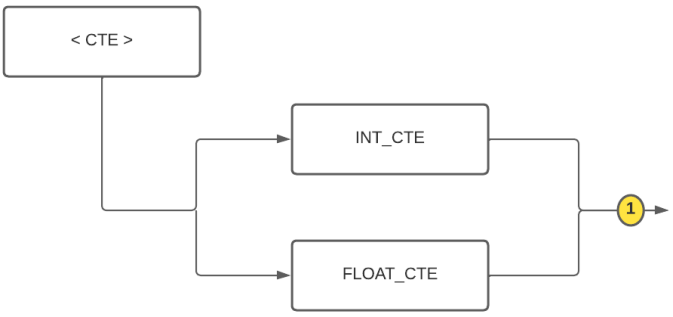
1. hacemos doble pop de tipo, de memoria y de operandos, checamos cubo semántico y generamos cuádruplo
2. Si es un * o un / hacemos push a pilas de operadores



1. Se mete a la pila de operadores.
2. Se hace pop a la pila de operadores



1. Hacemos doble pop de tipo, de memoria y de operandos, checamos cubo semántico y generamos cuádruplo
2. Si es un * o un / hacemos push a pilas de operadores



1. Generamos espacio de memoria para cte

- Tabla de consideraciones semánticas (combinaciones factibles y errores de tipo)

Operando 1	Operador	Operando 2	Resultado
Entero	{+, -, *, /, =}	Entero	Entero
Entero	{==, >, <, <=, >=, !=}	Entero	Booleano
Entero	{+, -, *, / }	Float	Float
Entero	{ = }	Float	Entero
Entero	{==, >, <, <=, >=, !=}	Float	Bool
Entero	{+, -, *, /, =, >, <, <=, >=, !=}	String	*Error*
Entero	==	String	Booleano
Float	{+, -, *, /, =}	Entero	Float
Float	{==, >, <, <=, >=, !=}	Entero	Booleano
Float	{+, -, *, /, =}	Float	Float
Float	{==, >, <, <=, >=, !=}	Float	Booleano
Float	{+, -, *, /, =, ==, >, <, <=, >=, !=}	String	*Error*
String	{+, -, *, /, =, ==, >, <, <=, >=, !=}	Entero	*Error*
String	{+, -, *, /, =, ==, >, <, <=, >=, !=}	Float	*Error*
String	{+, -, *, /, =, ==, >, <, <=, >=, !=}	String	*Error*

- Descripción detallada del proceso de administración de memoria usado en la compilación

  Para la memoria se utilizaron variables de límites y de inicio que sirven para delimitar el espacio dedicado a cada tipo de variable. Cada vez que se declaraba una variable o se generaba una temporal se contaba con contador para cada intervalo que se actualizaba.

  Para los cuadruplos se definió una clase que tiene 4 atributos:
  - (Operador, Operando1, Operando2, Resultado)

Para el cubo semántico se hizo una función que recibe 3 parámetros definidos en la operación y regresa el tipo esperado.

Para la tabla de variables y el directorio de funciones utilizamos diccionarios nativos de python por su eficiencia en las búsquedas.

DESCRIPCIÓN DE LA MÁQUINA VIRTUAL

- En el desarrollo de este proyecto no se alcanzó a desarrollar.

PRUEBAS DEL FUNCIONAMIENTO DEL LENGUAJE

- Incluir pruebas que comprueben el funcionamiento del proyecto
    - Pruebas de precedencia de operadores

```
program PRUEBA;
main () {
     entero cont = 2 / 3 + 8 * ( 2 + 4 ) ;
     }
```

```
entero
entero
1500 <class 'int'>
operadores
[]
operandos
[]
cuads:
[('/', 2, 3, 3000)]
[('+', 2, 4, 3001)]
[('*', 8, 3001, 3002)]
[('+', 3000, 3002, 3003)]
[('=', 3003, '', 1500)]
Tipos:
[]
Saltos
[]
```

    - Prueba de condicionales

```
program PRUEBA;

main () {
     entero cont = 5;
     entero cont2 = 3;
     if( cont > cont2 )
     then{
```

```
1500 <class 'int'>
1501 <class 'int'>
1502 <class 'int'>
2000 <class 'int'>
operadores
[]
operandos
```

```
      entero cont3 = 12;
      }
      else{
      float cont4 = 15;
      }

      }
```

```
[]
cuads:
[('=', 5, '', 1500)]
[('=', 3, '', 1501)]
[('>', 1500, 1501, 4500)]
('GOTO', '', '', 6)
[('=', 12, '', 1502)]
('GOTOF', 4500, '', 7)
[('=', 15, '', 2000)]
Tipos:
[]
Saltos
[]
```

- Prueba de repetición

```
program PRUEBA;

main () {

      while( 5 > 2 )
      do
      {
      entero cont = 2 / 3 + 8 * (
2 + 4 ) ;
      }
      }
```

```
entero
entero
1500 <class 'int'>
operadores
[]
operandos
[]
cuads:
[('>', 5, 2, 4500)]
('GOTOF', 4500, '', 8)
[('/', 2, 3, 3000)]
[('+', 2, 4, 3001)]
[('*', 8, 3001, 3002)]
[('+', 3000, 3002, 3003)]
[('=', 3003, '', 1500)]
[('GOTO', '', '', 0)]
Tipos:
[]
Saltos
[]
```

- Prueba de funciones

```
program PRUEBA;
function entero nom (entero :
```

```
tipoact: entero
nombfunc: nom
```

```
uno, float : dos);
      {
      if( 5 > 2 )
      then{
            entero cont3 = 12;
            float cont5 = 90;
            cont3 = 5;
      }
      else{
            cont3 = 15;
      }
      write();
      }

main () {
      entero cont = 5;
      entero cont2 = 3;
      while( 5 > 2 )
      do
      {
      entero cont = 2 / 3 + 8 * ( 
2 + 4 ) ;
      cont2 = 5*2;
      }
      }
```

```
Se agregó con éxito
llego a func2
tipoact: float
nombfunc: nom
Variable ya existe
Se agregó con éxito
llego a func2
1501 <class 'int'>
2001 <class 'int'>
1501 <class 'int'>
1501 <class 'int'>
1502 <class 'int'>
1503 <class 'int'>
Variable ya existe
entero
entero
1504 <class 'int'>
entero
1503 <class 'int'>
operadores
[]
operandos
[]
cuads:
[('>', 5, 2, 4500)]
('GOTO', '', '', 6)
[('=', 12, '', 1501)]
[('=', 90, '', 2001)]
[('=', 5, '', 1501)]
('GOTOF', 4500, '', 7)
[('=', 15, '', 1501)]
[('PRINT', '', '', '')]
[('ENDFUNC', '', '', '')]
[('=', 5, '', 1502)]
[('=', 3, '', 1503)]
[('>', 5, 2, 4501)]
('GOTOF', 4501, '', 21)
[('/', 2, 3, 3000)]
[('+', 2, 4, 3001)]
[('*', 8, 3001, 3002)]
[('+', 3000, 3002, 3003)]
[('=', 3003, '', 1504)]
```

	[('*', 5, 2, 3004)] [('=', 3004, '', 1503)] [('GOTO', '', '', 11)] Tipos: [] Saltos []

- Prueba de fibonacci

```
program PRUEBA;
main () {
      entero a = 0;
      entero b = 1;
      entero c = 0;
      entero x = 2;
      entero n = 10;

      if (10 == 0) then {
      write();
      }
      else{
      if (10 == 1) then{
            write();
      }
      else{
            while(x > 10)do{
            c = a + b;
            a = b;
            b = c;
            }
      write();
      }


      }
}
```

```
1500 <class 'int'>
1501 <class 'int'>
1502 <class 'int'>
1503 <class 'int'>
1504 <class 'int'>
1502 <class 'int'>
1500 <class 'int'>
1501 <class 'int'>
operadores
[]
operandos
[]
cuads:
[('=', 0, '', 1500)]
[('=', 1, '', 1501)]
[('=', 0, '', 1502)]
[('=', 2, '', 1503)]
[('=', 10, '', 1504)]
[('==', 10, 0, 4500)]
('GOTO', '', '', 9)
[('PRINT', '', '', '')]
('GOTOF', 4500, '', 21)
[('==', 10, 1, 4501)]
('GOTO', '', '', 13)
[('PRINT', '', '', '')]
('GOTOF', 4501, '', 21)
[('>', 1503, 10, 4502)]
('GOTOF', 4502, '', 20)
[('+', 1500, 1501, 3000)]
[('=', 3000, '', 1502)]
[('=', 1501, '', 1500)]
```

	[('=', 1502, '', 1501)] [('GOTO', '', '', 13)] [('PRINT', '', '', '')] Tipos: [] Saltos []