Tecnológico de Costa Rica

Centro Académico de Limón

Nombre del Curso: Compiladores e Intérpretes

Número de Portafolio: Número 2

Nombre del Profesor: Allan Rodríguez Dávila

Nombre del Estudiante: Jhonner Valverde Villachica

Número de Grupo: Grupo Número 60

Número de Semestre: Número 1

Año: 2025

Portafolio #2: Gramáticas

Gramática que permita un único main y cero o muchas funciones (main y funciones con una única expresión de asignación o creación)

<main> ::= 'void' 'main' '(' ')' '\\' <instrucciones> '/'

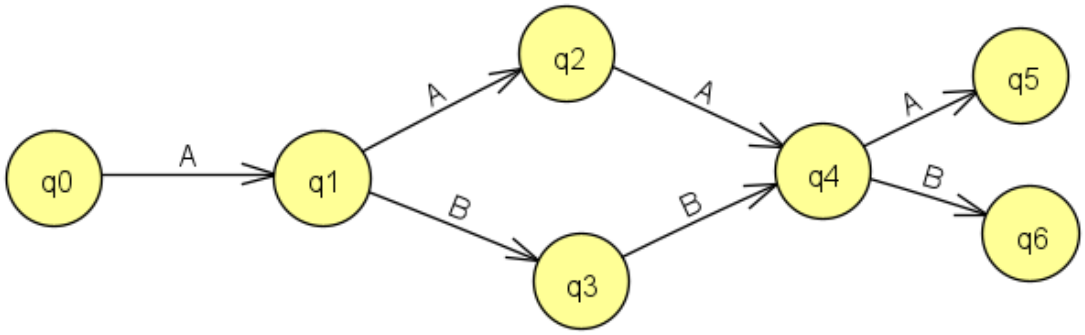
<instrucciones> ::= <instruccion> <instrucciones> |

<programa> ::= <main> | <main> <instrucciones>

Portafolio #2

- Desarrolle los AFN de los lenguajes denotados por las siguientes expresiones regulares y su tabla de transición:

a(a | b)a(a | b)

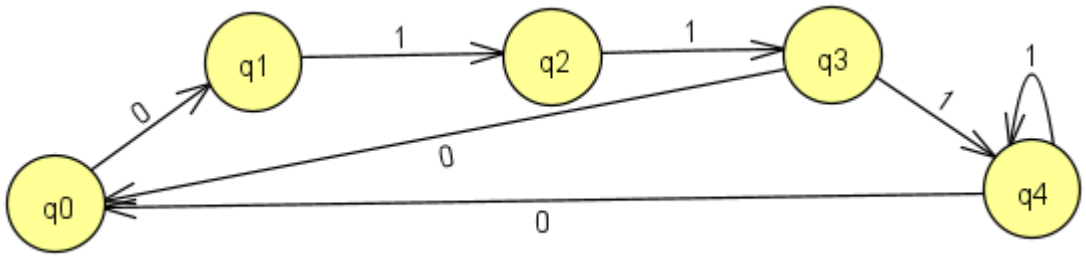


Estado	a	b	Ɛ
q0	{q1}	Ø	Ø
q1	{q2}	{q3}	Ø
q2	{q4}	Ø	Ø
q3	Ø	{q4}	Ø
q4	{q5}	{q6}	Ø
q5	Ø	Ø	Ø
q6	Ø	Ø	Ø

Portafolio #2

- Desarrolle los AFN de los lenguajes denotados por las siguientes expresiones regulares y su tabla de transición:

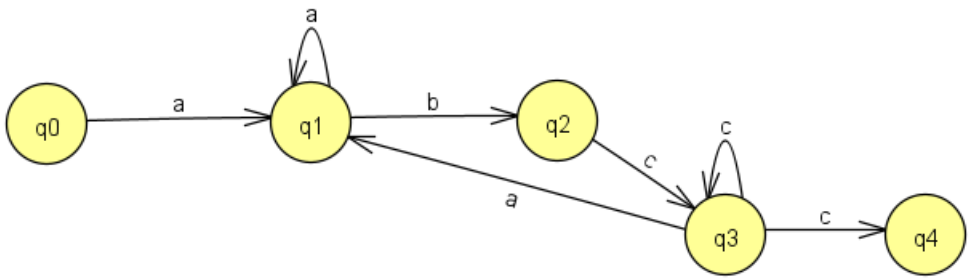
(011+)+



Estado	0	1	Ɛ
q0	{q1}	Ø	Ø
q1	Ø	{q2}	Ø
q2	Ø	{q3}	Ø
q3	{q0}	{q4}	Ø
q4	{q0}	{q4}	Ø

Portafolio 2, #3.1

- Desarrolle los **AFN** de los lenguajes denotados por las siguientes expresiones regulares:
  - a(a | bc+)*c



Portafolio 2, #3.2

- Desarrolle las **tablas de transición** de los AFN de los lenguajes denotados por las siguientes expresiones regulares:
  - a(a | bc+)*c

Estado	a	b	c	Ɛ
q0	{q1}	Ø	Ø	Ø
q1	{q1}	{q2}	Ø	Ø
q2	Ø	Ø	{q3}	Ø
q3	{q1}	Ø	{q3, q4}	Ø
q4	Ø	Ø	Ø	Ø

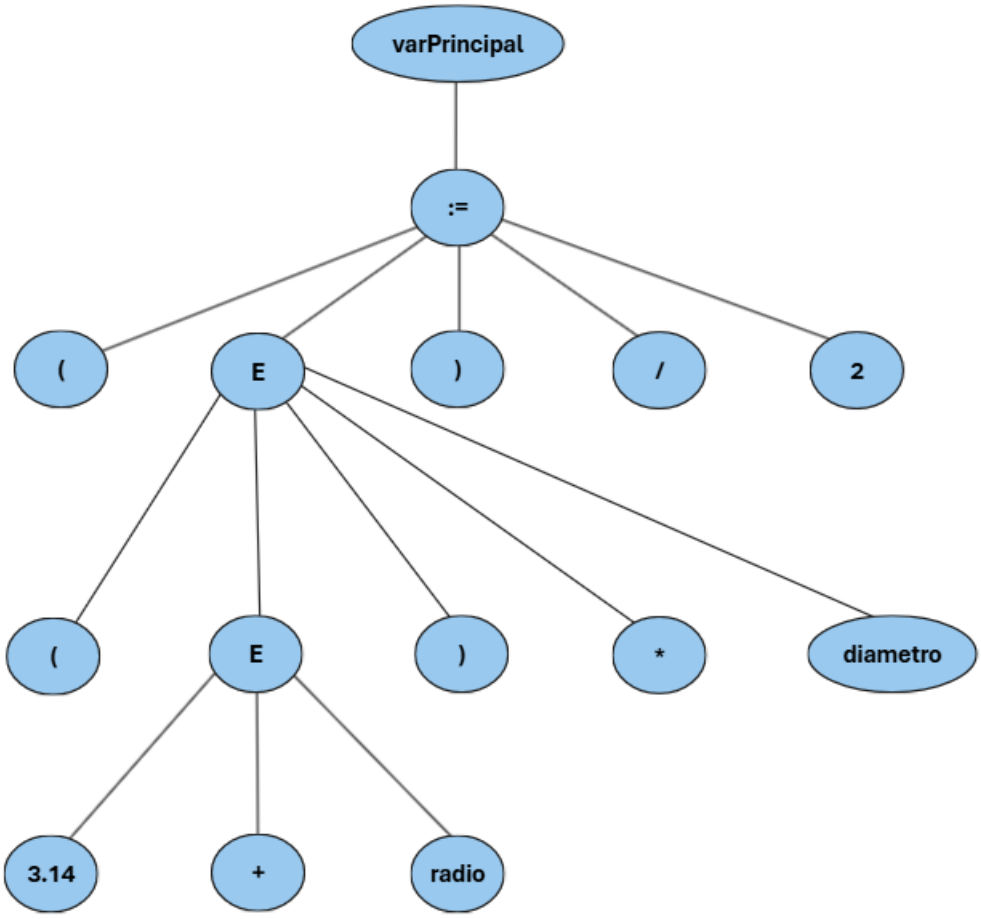
Portafolio 2, #3.3

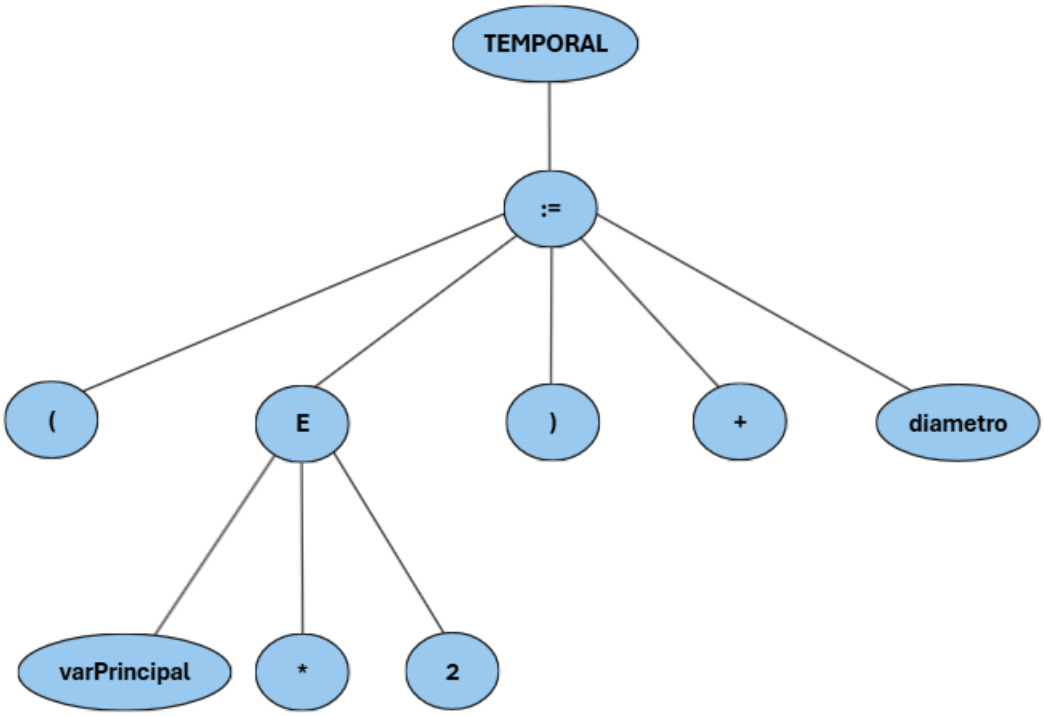
- Desarrolle los **AFD** de los lenguajes denotados por las siguientes expresiones regulares(con **tablas Dtran**):
  - a(a | bc+)*c

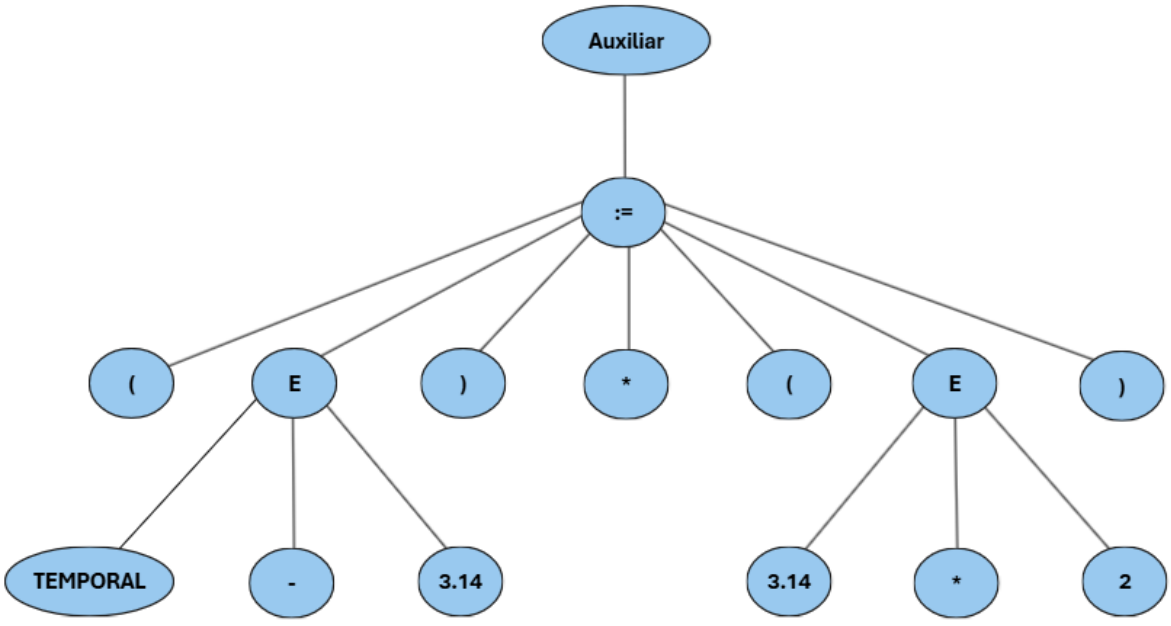
Estado AFN	Estado AFD	a	b	c
{q0}	A	{q1}	Ø	Ø
{q1}	B	{q1}	{q2}	Ø
{q2}	C	Ø	Ø	q3
{q3}	D	{q1}	Ø	{q3, q4}
{q3, q4}	E	{q1}	Ø	{q3, q4}

Portafolio #2 Construcción AS

- `varPrincipal:=((3.14+radio)*diametro)/2`
- `TEMPORAL:=(varPrincipal*2)+diametro`
- `Auxiliar:=(TEMPORAL-3.14)*(3.14*2)`




TEMPORAL
:=
(
E
)
+
diametro
varPrincipal
*
2


Auxiliar
:=
(
E
)
*
(
E
)
TEMPORAL
-
3.14
3.14
*
2

Portafolio #2

- Investigar técnicas de automatización de autómatas:
  - Minimización de Estados
  - Combinación de Estados Equivalentes
  - Otros

1. **Minimización de Estados**
   - Partición inicial.
   - Partición de nuevos conjuntos.
   - Verificar la nueva partición.
   - Eliminar los estados que se encuentren en el mismo grupo.
   - Remover estados sumidero y no alcanzables desde símbolo inicial.

   Enlace: Minimización de un autómata determinista

2. **Combinación de Estados Equivalentes**
   Dado un autómata finito no determinista cualquiera, se construye uno determinista equivalente en el que los estados son subconjuntos de estados del autómata de partida.

   Enlace: Equivalencias automatas deterministas no deterministas

3. **Máquina de Estado Mealy**
   Cuando las salidas dependen tanto de las entradas como de los estados actuales, el FSM puede denominarse máquina de estados harinosos. El siguiente diagrama es el diagrama de bloques de la máquina de estado mealy. El diagrama de bloques de la máquina de estado mealy consta de dos partes: la lógica combinacional y la memoria. La memoria de la máquina puede utilizarse para proporcionar algunas de las salidas anteriores como entradas de la lógica combinacional.

4. **Máquina de Estado de Moore**
   Cuando las salidas dependen de los estados actuales, el FSM puede denominarse Máquina de estado de Moore. El Diagrama de bloques de la máquina de

estado de Moore se muestra a continuación. El diagrama de bloques de la máquina de estado de Moore consta de dos partes: la lógica combinacional y la memoria.

Enlace: Máquina De Estado De Mealy Y Máquina De Estado De Moore

Portafolio #2: Resumen

- Lectura y resumen
  - Gramáticas Libres de contexto:
    - Definición
    - Notación
    - Derivaciones
    - Gramática Pascal, C

1. **Definición**
   Una gramática libre de contexto consta de terminales, no terminales, un símbolo de inicio y producciones.
   - Los terminales son los símbolos básicos a partir de los cuales se forman las cadenas. El término "nombre de token" es sinónimo de "terminal" y con frecuencia usaremos la palabra "token" para terminal cuando quede claro que nos referimos únicamente al nombre del token.
   - Los no terminales son variables sintácticas que denotan conjuntos de cadenas. Los conjuntos de cadenas denotados por no terminales ayudan a definir el lenguaje generado por la gramática. Los no terminales imponen una estructura jerárquica en el lenguaje que es clave para el análisis y la traducción de la sintaxis.
   - En una gramática, un no terminal se distingue como el símbolo de inicio, y el conjunto de cadenas que denota es el lenguaje generado por la gramática. Convencionalmente, las producciones para el símbolo de inicio se enumeran primero.
   - Las producciones de una gramática especifican la manera en que los terminales y los no terminales se pueden combinar para formar cadenas.

2. **Notación**
   Para evitar tener que indicar siempre que "estos son los terminales", "estos son los no terminales", etc., se utilizarán las siguientes convenciones de notación para gramáticas.
   - **Terminales:**
     a) Letras minúsculas al principio del alfabeto.
     b) Símbolos de operador.

c) Símbolos de puntuación.
d) Dígitos numéricos.
e) Cadenas en negrita que representan un único símbolo terminal.

- **No Terminales:**
  a) Letras mayúsculas al principio del alfabeto.
  b) La letra S que cuando aparece, suele ser el símbolo de inicio.
  c) Nombres en minúscula y cursiva.

3. Derivaciones

La construcción de un árbol de análisis sintáctico puede precisarse mediante una perspectiva derivacional, en la que las producciones se tratan como reglas de reescritura. Comenzando con el símbolo de inicio, cada paso de reescritura reemplaza un no terminal por el cuerpo de una de sus producciones. Esta perspectiva derivacional corresponde a la construcción descendente de un árbol de análisis sintáctico, pero la precisión que ofrecen las derivaciones será especialmente útil cuando se analice el análisis sintáctico ascendente.

Enlace: Lectura 3

4. Gramáticas

- **Gramática de Pascal**
  - Alternativa doble ::= **if** <Condicion> **then** <Sentencia> **else** <Sentencia>
  - Alternativa multiple ::= **case** <Expresion de Tipo ordinal> **of** <Caso>{; <Caso>} **end**
    Alternativa multiple ::= **case** <Expresion de Tipo ordinal> **of** <Caso>{; <Caso>} **else** <Sentencia> **end** ‡
  - Alternativa simple ::= **if** <Condicion> **then** <Sentencia>
  - Bucle con numero fijo de iteraciones ::= **for** <Sentencia de asignacion> **to** <Expresion de Tipo ordinal> **do** <Sentencia>
    Bucle con numero fijo de iteraciones ::= **for** <Sentencia de asignacion> **downto** <Expresion de Tipo ordinal> **do** <Sentencia> ‡
  - Bucle con salida al final ::= **repeat** <Sentencia>{; <Sentencia>} **until** <Condicion>
  - Bucle con salida al principio ::= **while** <Condicion> **do** <Sentencia>
  - Cabecera de funcion ::= **function** <Identificador de funcion> **:** <Identificador de Tipo elemental> ;
    Cabecera de funcion ::= **function** <Identificador de funcion> **(** <Parametros formales por valor>{; <Parametros formales por valor>} **) :** <Identificador de Tipo elemental> ;
  - Cabecera de procedimiento ::= **procedure** <Identificador de procedimiento> ;
    Cabecera de procedimiento ::= **procedure** <Identificador de procedimiento> **(** <Parametros formales>{; <Parametros formales>} **) ;**
  - Cabecera de programa ::= **program** <Identificador de programa> ;
    Cabecera de programa ::= **program** <Identificador de programa> **(** <Lista de dispositivos> **);**
  - Cabecera de unidad ::= **unit** <Identificador de unidad> ;

Enlace: Sintaxis Backus-Naur de Pascal

- **Gramática de C**

Tokens

token:

keyword

identifier

constant

string-literal

punctuator

preprocessing-token:

header-name

identifier

pp-number

character-constant

string-literal

punctuator

cada carácter que no sea un espacio en blanco que no pueda ser uno de los anteriores

Palabras clave

keyword: uno de

auto **break** **case** **char** **const** **continue**

default **do** **double** **else** **enum** **extern**

float **for** **goto** **if** **inline** **int** **long**

register **restrict** **return** **short** **signed**

sizeof **static** **struct** **switch** **typedef** **union**

unsigned **void** **volatile** **while** **_Alignas**

_Alignof **_Atomic** **_Bool** **_Complex** **_Generic**

_Imaginary **_Noreturn** **_Static_assert**

_Thread_local

Para una lista de otras palabras clave específicas de Microsoft, consulte Palabras clave de C.

Identificadores

identifier:

identifier-nondigit

identifier *identifier-nondigit*

identifier *digit*

identifier-nondigit:

nondigit

universal-character-name

otros caracteres definidos por la implementación

nondigit: uno de

_ **a** **b** **c** **d** **e** **f** **g** **h** **i** **j** **k** **l** **m**

n **o** **p** **q** **r** **s** **t** **u** **v** **w** **x** **y** **z**

A **B** **C** **D** **E** **F** **G** **H** **I** **J** **K** **L** **M**

N **O** **P** **Q** **R** **S** **T** **U** **V** **W** **X** **Y** **Z**

digit: uno de

0 **1** **2** **3** **4** **5** **6** **7** **8** **9**

universal-character-name:

\u *hex-quad*

\U *hex-quad* *hex-quad*

hex-quad:

hexadecimal-digit *hexadecimal-digit* *hexadecimal-digit* *hexadecimal-digit*

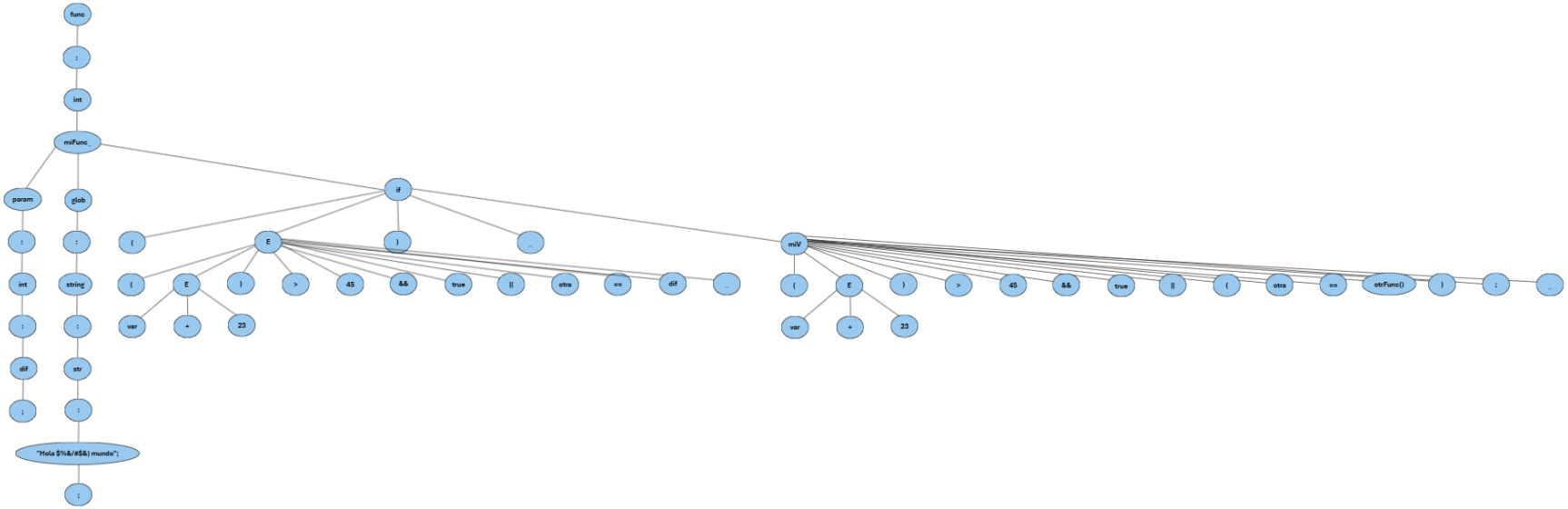
Enlaces: Gramática léxica de C | Microsoft Learn - Gramática de la estructura de la frase | Microsoft Learn

Portafolio #2: AS

- Gramática de curso

```
func:int:main _ miFunc(hola,'a’); return; _

func:int:miFunc_
        param:int:dif;
        loc:int:a;
        param:char:otra;
        glob:string:str="Hola $%&/#$&) mundo";
        loc:int:var=34 + id ^ miFunc2 (func3(34,id));
        if ((var+23) > 45 && true || otra == dif)_
                miV = (var+23) > 45 && true || (otra == otrFunc()) ;
        _
        for:var:in:range(5) _ print(“Hola mundo”); _
        return:3+4;
```


func
:
int
miFunc_
param
glob
:
:
int
string
:
:
dif
str
:
:
"Hola $%&/#$&) mundo";
;
if
(
E
)
_
(
E
)
>
45
&&
true
||
otra
==
dif
_
var
+
23
miV
(
E
)
>
45
&&
true
||
(
otra
==
otrFunc()
)
:
_
var
+
23