TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

ENERO JUNIO 2017

CARRERA: ING. EN SISTEMAS COMPUTACIONALES

MATERIA Y SERIE:
LENGUAJES Y AUTÓMATAS I
SCD-1015 SC6A

GRUPO: A

TÍTULO: ANALIZADOR SINTÁCTICO Y GRAMÁTICA

UNIDADES:
V Y VI

TEMAS:
ANALIZADOR SINTÁCTICO, GRAMÁTICA

NOMBRE DE MAESTRO: ESTRADA PEÑA ERASMO

AYALA SANDOVAL JESUS EDUARDO #14211403

FECHA DE ENTREGA: 30/05/2017

Introducción

Parte fundamental de un lenguaje de programación, es su gramática. Esta define todas las reglas que debe seguir la manera en que podemos escribir el código que será utilizado. Nuestro lenguaje debe contener gramática definida para todos los casos de uso que pueda tener. El analizador sintáctico es el encargado de leer el código fuente e interpretarlo de acuerdo a la gramática definida, el analizador sintáctico trabaja junto al analizador léxico para leer y formar las expresiones deseadas.

Gramática

Las gramáticas son descripciones de las sentencias de los lenguajes, establecen la estructura que deben tener las sentencias para que estén bien formadas y para que pueda entenderse su significado.

Las gramáticas formales son descripciones estructurales de las sentencias de los lenguajes, tanto formales (lenguajes de programación), como naturales (humanos). En este último caso, la gramática se ocupa de la sintaxis (es decir la forma que deben tener las sentencias).

Las gramáticas permiten describir de forma intencional a los lenguajes; proporcionan reglas para la estructura de las frases y su significado.

Informalmente podemos decir que una gramática es un modelo matemático que consta de:

- Un alfabeto, llamado alfabeto de elementos terminales que representa el conjunto de letras que tendrán las palabras del lenguaje que genera esa gramática.
- Un conjunto de símbolos especiales, denominados no terminales, que son elementos auxiliares y permiten representar estados intermedios antes de llegar al de la generación de las palabras del lenguaje.
- Un símbolo inicial del que se partirá para la obtención de cualquiera de las palabras del lenguaje, denominado cabeza del lenguaje, que es uno de los elementos no terminales.
- Un conjunto de producciones o reglas gramaticales que permitirán realizar las transformaciones desde los símbolos no terminales a las palabras del lenguaje.

Analizador Sintáctico

La función del analizador sintáctico es la de analizar el texto de entrada de acuerdo a una gramática dada. En caso de que el texto ingresado sea válido, suministrar el árbol sintáctico que lo reconoce.

El analizado sintáctico reconoce la secuencia de tokens suministrada por el analizador léxico, siguiendo las reglas ya establecidas.

Además, algunas de las funciones del analizador sintáctico son:

- Acceder a la tabla de símbolos
- Generar errores cuando se producen
- Recuperación de errores

La gramatica que acepta el analizador sintáctico es gramatica de contexto libre. La cual tiene como elementos

- No Terminales -> Son los que pueden ser derivados a mas producciones
- Terminales -> No es posible realizar derivaciones
- Reglas de producción -> Son las reglas que sigue la gramática

Gramática de programa

archivo_compilado -> paquete libreria clase_interface_declaracion

paquete-> K_PACKAGE qualified_name EOL

libreria->(librerias'|vacio)
librerias'-> librerias(librerías'|vacio)
librerias-> K_REQUIRE qualified_name EOL

clase_interface_declaracion-> (clase_declaration|interface_declaration)

clase_declaration-> K_CLASS Identificador extend implement bodyclase K_END

extend-> (extends|vacio)
extends-> K_EXTENDS Identificador

implement-> (implements|vacio)
implements -> K_IMPLEMENTS qualified_name (implements'|vacio)
implements'->im (implements'|vacio)
im->(COMA qualified_name)

interface_declaration-> K_INTERFACE Identificador bodyclase K_END

bodyclase->(bodyclass|vacio)
bodyclass-->bodyuses(bodyclass|vacio)
bodyuses->(acciones| definir_variables| definir_arreglo| definir_propiedad| ciclos| clase_interface_declaracion| metodos| constructor)

acciones->
(imprimir|asignacion_especial|retorno|conversion_variable|accion_sistema|condicional|invocar_metodo_funcion|crear_objeto|referencia_objeto|inspeccionar_objeto|matematica_especial|accion_num|ajuste_array|ordenar_arreglo|busqueda_array|split|manejo_archivos|accion_arreglo|union|asignación)

imprimir-> K_IMPRIMIR valor EOL

asignacion_especial-> ASIGNACION_ESP LPAR bodyasign RPAR EOL
bodyasign->identificadores COMA valor

retorno-> K_RETORNO EOL

retorno_valor-> K_RETORNO valor EOL

conversion_variable-> CONVERSION LPAR identificadores RPAR EOL

accion_sistema-> ACCIONSYS EOL

condicional-> LLAIZQ bodyexpresiones LLADER K_IF LPAR condicion_exp RPAR
(elseb|vacio)
bodyexpresiones-> (bodyexp|vacio)
bodyexp-> bodyexp'(bodyexp|vacio)
bodyexp'->(acciones|ciclos|definir_var_local|retorno_valor)

ciclos-> (ciclo_times|ciclo_each)

ciclo_times-> DecimalLiteral K_TIMES K_DO LLAIZQ bodyexpresiones LLADER

ciclo_each-> identificadores K_EACH K_DO DELIM var_local DELIM LLAIZQ
bodyexpresiones LLADER

condicion_exp-> condicion cond
cond->(cond'|vacio)
cond'->OP_LOG condición_exp
condicion-> valor OP_REL valor

elseb-> K_ELSE LLAIZQ bodyexpresiones LLADER

definir_var_local-> var_local LPAR body_var_local RPAR EOL
body_var_local-> TIPO COMA (valor|vacio)

invocar_metodo_funcion-> K_INVOKE qualified_name LPAR (parametros|vacio) RPAR EOL

parametros-> valor param'
param'->(secuencia|vacio)
secuencia-> COMA parametros

crear_objeto-> qualified_name PUNTO K_NEW LPAR (parametros|vacio) RPAR EOL

referencia_objeto-> REFERENCIA LPAR identificadores RPAR EOL

inspeccionar_objeto-> K_INSPECCIONAR LPAR identificadores RPAR EOL

usar_metodo-> qualified_name LPAR (parametros|vacio) RPAR

matematica_especial-> MATEMATICA LPAR identificadores COMA valor RPAR EOL

accion_num-> NUM LPAR identificadores COMA valor RPAR EOL

ajuste_array-> K_RESIZE LPAR identificadores RPAR EOL

ordenar_arreglo->K_ORDENAR MODO_ORD LPAR identificadores RPAR EOL

busqueda_array-> identificadores K_WHERE identificadores TIPOBUSQUEDA valor EOL

split-> K_SPLIT LPAR string RPAR identificadores EOL

manejo_archivos-> K_BEGIN K_DIR LPAR manejo RPAR EOL
manejo-> ACCIONARCHIVO COMA string

accion_arreglo->ACCIONARREGLO LPAR identificadores COMA valor RPAR EOL

union->string K_UNION LPAR identificadores RPAR string EOL

asignacion-> identificadores ASIGNACION expresion EOL

expresion->valor expresion'
expresion'->(OP_ARI expresion)|vacio

definir_variables-> (definir_variable|definir_var_local)

definir_variable-> K_DEF K_VAR Identificador LPAR var_exp EOL

var_exp-> TIPO COMA VISIBILIDAD COMA (MODIFICADOR|vacio) COMA (valor|vacio)
RPAR

definir_arreglo-> K_DEF K_ARRAY identificadores LPAR DecimalLiteral RPAR LLAIZQ
(exp_arreglo|vacio) LLADER EOL
exp_arreglo-> valor exp_arreglo'
exp_arreglo'->(COMA exp_arreglo)|vacio

definir_propiedad-> K_DEF K_PROPIEDAD Identificador LPAR Identificador RPAR LLAIZQ
body_propiedad LLADER

body_propiedad-> get (set|vacio)
get-> K_GET LLAIZQ Identificador LLADER
set-> K_SET LLAIZQ K_VALUE LLADER

constructor-> K_DEF K_INITIALIZE LPAR param_tipo RPAR LLAIZQ bodyexpresiones LLADER

param_tipo->(parametros_tipos|vacio)
parametros_tipos-> TIPO valor param_t'
param_t'->(COMA parametros_tipos)|vacio

metodos-> (funcion|metodo)

funcion-> K_DEF K_FUNC Identificador LPAR param_tipo RPAR LLAIZQ bodyexpresiones LLADER

metodo-> K_DEF K_VOID Identificador LPAR param_tipo RPAR LLAIZQ bodyexpresiones LLADER

valor-> (char|string|DecimalLiteral|booleano|var_local|usar_metodo|qualified_name)

var_local-> DOUBLEDOT Identificador

comentario-> # CaracteresNoSaltoLinea

identificador->(Identificador|var_local)

Programa

```
grammar Compilador;
options {
        backtrack=true;
  language = Ruby;
  output = AST;
        }
archivo_compilado
        : paquete librerias* clase_interface_declaracion
        ;
paquete
        : K_PACKAGE qualified_name EOL
        ;
librerias
        : K_REQUIRE qualified_name EOL
        ;
clase_interface_declaracion
        :(clase_declaration|interface_declaration)
        ;
clase_declaration
        : K_CLASS Identificador extends? implements? bodyclass K_END
        ;
extends
        :K_EXTENDS Identificador
        ;
implements
        : K_IMPLEMENTS qualified_name (COMA qualified_name)*
        ;
interface_declaration
        : K_INTERFACE Identificador bodyclass K_END
        ;
bodyclass
        : (acciones
        | definir_variables
        | definir_arreglo
        | definir_propiedad
        | ciclos
        | clase_interface_declaracion
        | metodos
        | constructor)*
        ;
acciones
        : imprimir
```

```
        |asignacion_especial
        |retorno
        |conversion_variable
        |accion_sistema
        |condicional
        |invocar_metodo_funcion
        |crear_objeto
        |referencia_objeto
        |inspeccionar_objeto
        |matematica_especial
        |accion_num
        |ajuste_array
        |ordenar_arreglo
        |busqueda_array
        |split
        |manejo_archivos
        |accion_arreglo
        |union
        |asignacion
        ;
imprimir
        : K_IMPRIMIR valor EOL
        ;
asignacion_especial
        : K_ASIGNACION LPAR (Identificador|var_local) COMA valor RPAR EOL
        ;
retorno
        : K_RETORNO EOL
        ;
retorno_valor
        : K_RETORNO valor EOL
        ;
conversion_variable
        : K_CONVERSION LPAR (Identificador|var_local) RPAR EOL
        ;
accion_sistema
        : K_ACCIONSYS EOL
        ;
condicional
        : LLAIZQ bodyexp LLADER K_IF LPAR condicion_exp RPAR elseb?
        ;
bodyexp
        :(acciones|ciclos|definir_var_local|retorno_valor)*
        ;
```

```
ciclos
        : (ciclo_times|ciclo_each)
        ;
ciclo_times
        : DecimalLiteral K_TIMES K_DO LLAIZQ bodyexp LLADER
        ;
ciclo_each
        : (Identificador|var_local) K_EACH K_DO DELIM var_local DELIM LLAIZQ bodyexp
LLADER
        ;
condicion_exp
        : condicion (OP_LOG condicion)*
        ;
condicion
        : valor OP_REL valor
        ;
elseb
        : K_ELSE LLAIZQ bodyexp LLADER
        ;
definir_var_local
        : var_local LPAR body_var_local RPAR EOL
        ;
body_var_local
        : TIPO COMA valor?
        ;
invocar_metodo_funcion
        : K_INVOKE qualified_name LPAR parametros? RPAR EOL
        ;
parametros
        : valor (COMA valor)*
        ;
crear_objeto
        : qualified_name PUNTO K_NEW LPAR parametros? RPAR EOL
        ;
referencia_objeto
        : K_REFERENCIA LPAR (Identificador|var_local) RPAR EOL
        ;
inspeccionar_objeto
        : K_INSPECCIONAR LPAR (Identificador|var_local) RPAR EOL
        ;
usar_metodo
        : qualified_name LPAR parametros? RPAR
        ;
matematica_especial
```

```
        : K_MATEMATICA LPAR (Identificador|var_local) COMA valor RPAR EOL
        ;
accion_num
        : K_NUM LPAR (Identificador|var_local) COMA valor RPAR EOL
        ;
ajuste_array
        : K_RESIZE LPAR (Identificador|var_local) RPAR EOL
        ;
ordenar_arreglo
        : K_ORDENAR ModoOrd LPAR (Identificador|var_local) RPAR EOL
        ;
busqueda_array
        : (Identificador|var_local) K_WHERE (Identificador|var_local) K_TIPOBUSQUEDA valor
EOL
        ;
split
        : K_SPLIT LPAR string RPAR (Identificador|var_local) EOL
        ;
manejo_archivos
        : K_BEGIN K_DIR LPAR manejo RPAR EOL
        ;
manejo
        : K_ACCIONARCHIVO COMA string COMA  Identificador
        ;
accion_arreglo
        : K_ACCIONARREGLO LPAR (Identificador|var_local) COMA valor RPAR EOL
        ;
union
        :string K_UNION LPAR (Identificador|var_local) RPAR string EOL
        ;
asignacion
        : (Identificador|var_local) ASIGNACION expresion EOL
        ;
expresion
        : valor (OP_ARI valor)*
        ;
definir_variables
        : (definir_variable|definir_var_local)
        ;
definir_variable
        : K_DEF K_VAR Identificador LPAR var_exp EOL
        ;
var_exp
        : TIPO COMA K_VISIBILIDAD COMA K_MODIFICADOR? COMA valor? RPAR
```

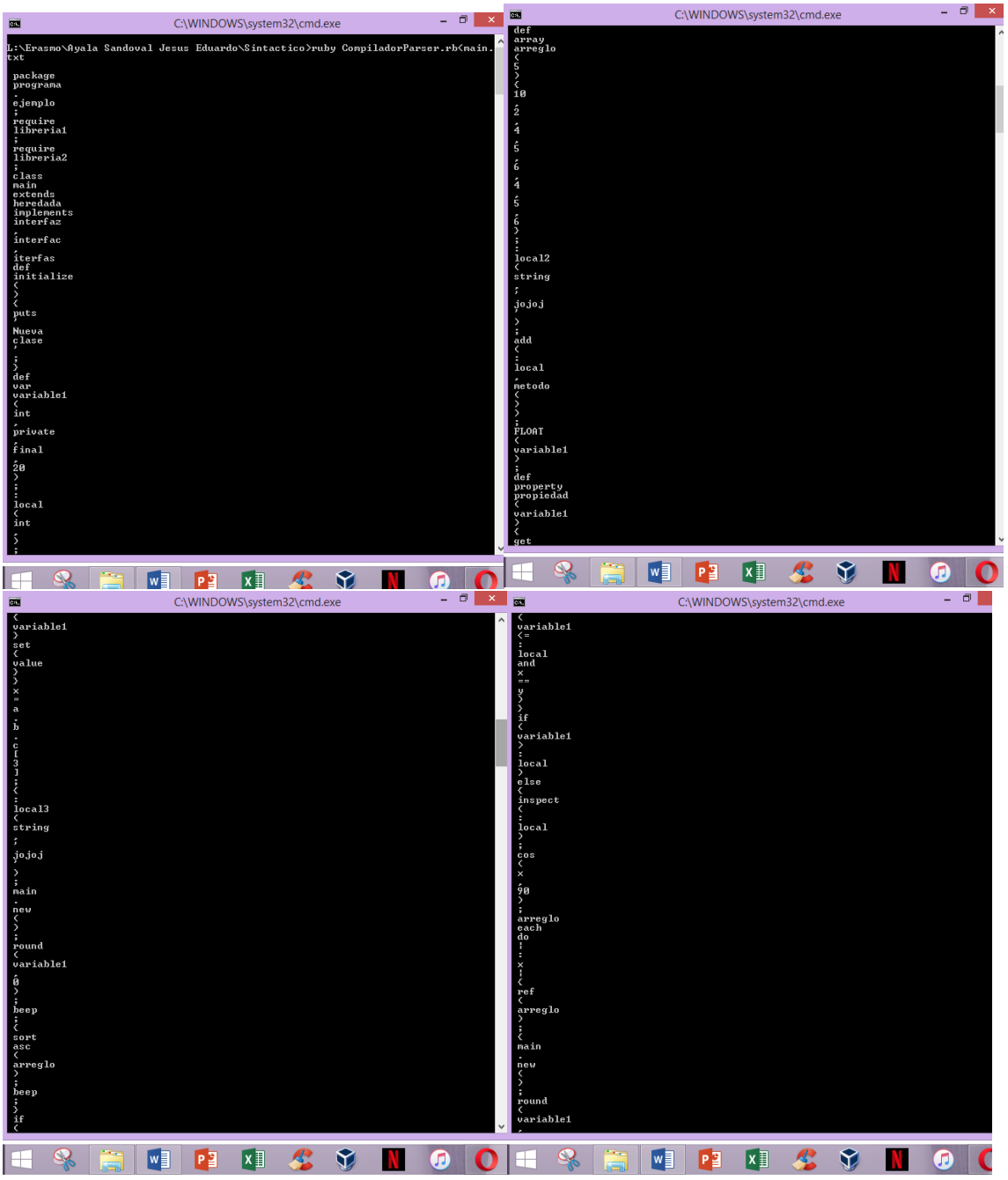
```
        ;
definir_arreglo
        : K_DEF K_ARRAY (Identificador|var_local) LPAR DecimalLiteral RPAR LLAIZQ
exp_arreglo LLADER EOL
        ;
exp_arreglo
        : valor (COMA valor)*
        ;
definir_propiedad
        : K_DEF K_PROPIEDAD Identificador LPAR Identificador RPAR LLAIZQ
body_propiedad LLADER
        ;
body_propiedad
                :  get set?
                ;
get
:       K_GET LLAIZQ Identificador LLADER
;
set
:       K_SET LLAIZQ K_VALUE LLADER
;
constructor
        : K_DEF K_INITIALIZE LPAR parametros_tipos? RPAR LLAIZQ bodyexp LLADER
        ;
parametros_tipos
        : TIPO valor (COMA TIPO valor)*
        ;
metodos
        :(funcion|metodo)
        ;
funcion
        : K_DEF K_FUNC Identificador LPAR parametros_tipos? RPAR LLAIZQ bodyexp
LLADER
        ;
metodo
        : K_DEF K_VOID Identificador LPAR parametros_tipos? RPAR LLAIZQ bodyexp
LLADER
        ;
```

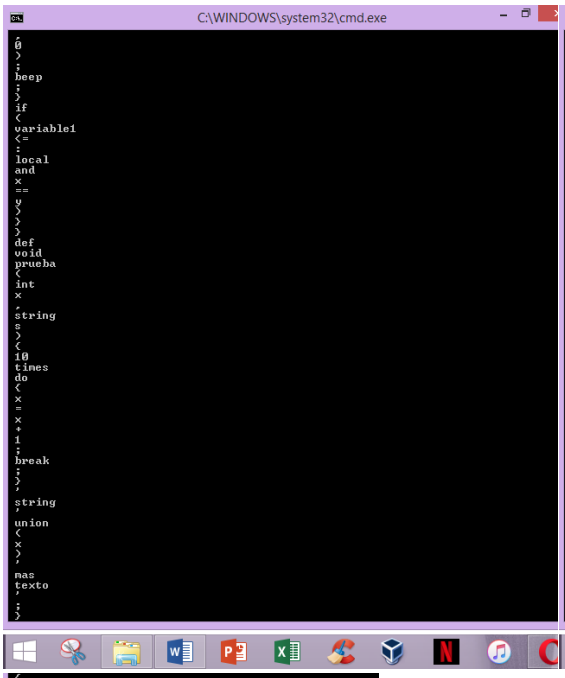
Archivo Fuente

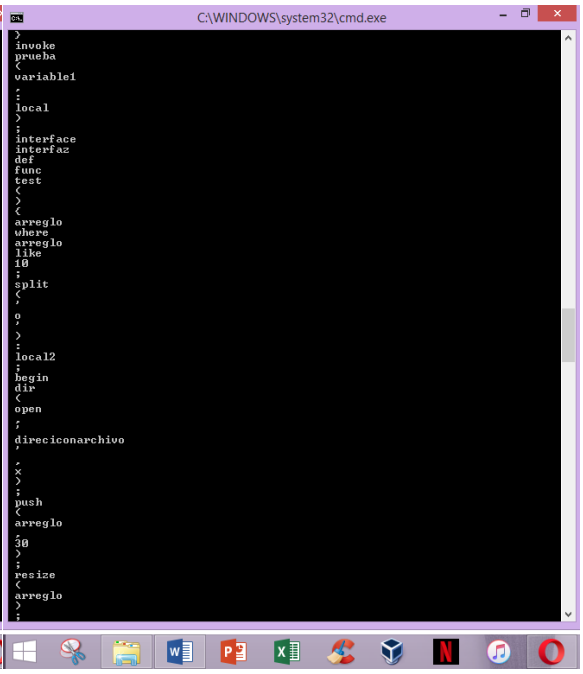
```
package programa.ejemplo;

#Esto es un comentario
require libreria1;
require libreria2;
class main extends heredada implements interfaz,interfac,iterfas
def initialize (){
	puts 'Nueva clase';
}
def var variable1 (int,private,final,20);
:local (int,);
def array arreglo(5){10,2,4,5,6,4,5,6};
:local2 (string,'jojoj');
add(:local,metodo());
FLOAT(variable1);
def property propiedad(variable1){
	get {variable1}
	set {value}
}
x=a.b.c[3];
#holabb
{
:local3 (string,'jojoj');
	main.new();
	round(variable1,0);
	beep;
	{
		sort asc (arreglo);
		beep;
	} if (variable1<=:local and x==y )
} if (variable1>:local)
else
{
	inspect(:local);
	cos(x,90);
	arreglo each do |:x|{
	ref(arreglo);
	{
		main.new();
		round(variable1,0);
```

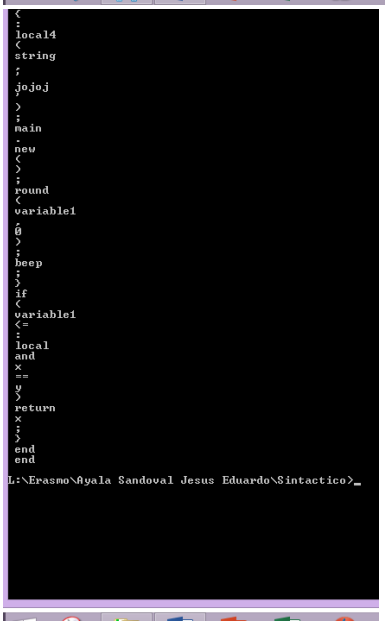
```
                beep;
        } if (variable1<=:local and x==y )
        }
}
def void prueba(int x, string s)
{
        10 times do {
        x=x+1;
        break;
        }
        'string' union (x) 'mas texto';
}
invoke prueba(variable1,:local);

interface interfaz
def func test(){
        arreglo where arreglo like 10;
        split('o') :local2;
        begin dir (open,'direciconarchivo',x);
        push (arreglo,30);
        resize(arreglo);
        {
        :local4 (string,'jojoj');
                main.new();
                round(variable1,0);
                beep;
        } if (variable1<=:local and x==y )
        return x;
}
end
end
```

Corrida Ejemplo




C:\WINDOWS\system32\cmd.exe
,
0
)
;
beep
;
)
if
(
variable1
<=
:
local
and
x
==
y
)
)
)
def
void
prueba
(
int
x
,
string
s
)
(
10
times
do
(
x
=
x
+
1
;
break
;
)
'
string
'
union
(
x
)
'
mas
texto
'
;
)


C:\WINDOWS\system32\cmd.exe
)
invoke
prueba
(
variable1
,
:
local
)
;
interface
interfaz
def
func
test
(
)
(
arreglo
where
arreglo
like
10
;
split
(
'
o
'
)
;
local2
;
begin
dir
(
open
;
'
direciconarchivo
'
,
x
)
;
push
(
arreglo
,
30
)
;
resize
(
arreglo
)
;


(
:
local4
(
string
;
'
jojoj
'
)
;
main
.
new
(
)
;
round
(
variable1
,
0
)
;
beep
;
)
if
(
variable1
<=
:
local
and
x
==
y
)
return
x
;
)
end
end
L:\Erasmo\Ayala Sandoval Jesus Eduardo\Sintactico>