UNIVERSIDAD POLITECNICA DE MADRID

**PRACTICA: ANALIZADOR LEXICO Y SINTACTICO**

Autores: Jiménez Pérez Juan, N:190204

Hernandez Pérez Jesus, N: 190295

Yanez Soffia Miguel, N: 190322

Grupo: 42

29 de noviembre del 2021

# Índice

Contenido

[Índice 2](#_Toc92537844)

[Diseño de la practica 3](#_Toc92537845)

[Consideraciones 3](#_Toc92537846)

[Analizador Léxico 3](#_Toc92537847)

[Tokens 3](#_Toc92537848)

[Autómata 3](#_Toc92537849)

[Gramática 4](#_Toc92537850)

[Acciones semánticas 4](#_Toc92537851)

[Error 5](#_Toc92537852)

[Analizador sintáctico 5](#_Toc92537853)

[Gramática 5](#_Toc92537854)

[Condiciones LL(1) 6](#_Toc92537855)

[Pseudo-Código 7](#_Toc92537856)

[Analizador Sintáctico 7](#_Toc92537857)

[Producciones 7](#_Toc92537858)

[Función Error 11](#_Toc92537859)

[Casos de prueba 11](#_Toc92537860)

[Caso 1 11](#_Toc92537861)

[Tokens: 12](#_Toc92537862)

[Parse: 13](#_Toc92537863)

[Caso 2 13](#_Toc92537864)

[Caso 3 14](#_Toc92537865)

[Caso 4 14](#_Toc92537866)

[Caso 5 15](#_Toc92537867)

[Caso 1 (Error) 16](#_Toc92537868)

[Caso 2 (Error) 17](#_Toc92537869)

[Caso 3 (Error) 18](#_Toc92537870)

[Caso 4 (Error) 18](#_Toc92537871)

[Caso 5 (Error) 19](#_Toc92537872)

# Diseño de la practica

Para el desarrollo del analizador léxico, hemos escogido el lenguaje de programación Java con la librería JFlex. Para el desarrollo del analizador sintáctico, hemos escogido el lenguaje de programación Java.

## Consideraciones

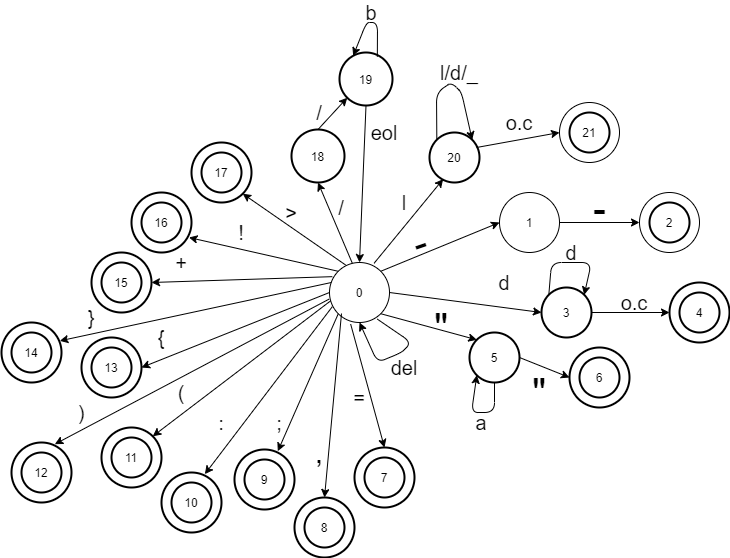
* Hemos utilizado los tokens obligatorios y de tokens de grupo la negación (!) y el mayor (>).
* Los comentarios se hacen con el doble barra diagonal (//) y finalizan con un salto de línea.
* Los enteros están en el rango [0 , 32767].
* Las cadenas están dentro de comillas.
* Un identificador es cualquier combinación de letras, números y barra baja (\_) que comience con una letra.

# Analizador Léxico

## Tokens

* Boolean: <boolean , >
* Break: <break, >
* Case: <case, >
* Function: <function, >
* If: <if, >
* Input: <input, >
* Int: <int, >
* Let: <let, >
* Print: <print, >
* Return: <return, >
* String: <string, >
* Switch: <switch, >
* Autodecremento: <opAutodecremento, >
* Constante entera: <entero, numero >
* Cadena: <cadena, lexema >
* Identificador: <id, numero >
* “=”: <opAsignacion, >
* “,”: <coma, >
* “;”: <puntoComa, >
* “:”: <dosPuntos, >
* “(“: <parentesisAbierto, >
* “)”: < parentesisCerrado, >
* “{“: <llaveAbierta, >
* “}”: <llaveCerrada,

## Autómata



## Gramática

l = letras minúsculas y mayúsculas

d = {0..9}

c = cualquier carácter

del = {espacio, eol, tab …}

a = c – { ”}

b = c – {eol}

o.c = otro caracter (diferente a letra y digito)

S 🡪 del S | l A | - C | d E | “ G | / K | = | , | ; | : | ( | ) | { | } | + | ! | >

A 🡪 l A | d A | \_ A | o.c

C 🡪 -

E 🡪 d E | o.c

G 🡪 a G | “

K 🡪 / L

L 🡪 b L | eol S

## Acciones semánticas

**0-0:** Leer.

**0-1:** Leer.

**1-2:** Leer; Gen\_Token(opAutodecremento, - ).

**0-3:** Numero = valor\_ASCII(d); Leer.

**3-3:** Numero = numero\*10 + valor\_ASCII(d); Leer.

**3-4:** Leer;IF(numero > 32767) THEN error(“Numero fuera de rango.”)

ELSE Gen\_Token(entero, numero).

**0-5:** Lexema := “;Leer

**5-5:** Lexema := concatenar(Lexema, a); Leer

**5-6:** Lexema := concatenar(Lexema, “); Leer;

If(Lexema.length<=64) Gen\_Token(cadena, lexema)

Else error (“cadena demasiado larga”)

**0-7:** Leer; Gen\_Token(opAsignacion, -)

**0-8:** Leer; Gen\_Token(coma, -)

**0-9:** Leer; Gen\_Token(puntoComa, -)

**0-10:** Leer; Gen\_Token(dosPuntos, -)

**0-11:** Leer; Gen\_Token(parentesisAbierto, -)

**0-12:** Leer; Gen\_Token(parentesisCerrado, -)

**0-13:** Leer; Gen\_Token(llaveAbierta, -)

**0-14:** Leer; Gen\_Token(llaveCerrada, -)

**0-15:** Leer; Gen\_Token(suma, -)

**0-16:** Leer; Gen\_Token(negacion, -)

**0-17:** Leer; Gen\_Token(mayor, -)

**0-18:** Leer;

**18-19:** Leer;

**19-19:** Leer;

**19-0:** Leer;

**0-20:** Lexema := l;Leer

**20-20:** Lexema := concatenar(Lexema, l/d,\_); Leer

**20-21:**

IF lexema == boolean THEN leer; Gen\_Token(Boolean, -)

ELSE IF lexema == break THEN leer; Gen\_Token(break, -)

ELSE IF lexema == case THEN leer; Gen\_Token(case, -)

ELSE IF lexema == function THEN leer; Gen\_Token(function, -)

ELSE IF lexema == if THEN leer; Gen\_Token(if, -)

ELSE IF lexema == input THEN leer; Gen\_Token(input, -)

ELSE IF lexema == int THEN leer; Gen\_Token(int, -)

ELSE IF lexema == print THEN leer; Gen\_Token(print, -)

ELSE IF lexema == return THEN leer; Gen\_Token(return, -)

ELSE IF lexema == string THEN leer; Gen\_Token(string, -)

ELSE IF lexema == switch THEN leer; Gen\_Token(switch, -)

ELSE IF Zona\_Declaracion = true THEN

{ p=buscar\_TS (lexema)

IF p=null THEN {p:=insertar\_TS(lexema) Gen\_Token (identificador, p) }

ELSE error (“identificador ya declarado”) }

ELSE { p=buscar\_TS (lexema)

IF p=null THEN error (“identificador NO declarado”)

ELSE Gen\_Token (identificador, p) }

## Error

Todas las transiciones no consideradas corresponden a casos de error.

# Analizador sintáctico

## Gramática

P -> B Z | F Z

B -> let T id; | if ( E ) S | switch ( Y ) { W } | S

T -> int | string | boolean

S -> idS’| print ( E ) ; | input ( id ); | return X;

S’ -> =E; | ( L ) ;

X -> E | lambda

C -> BC | lambda

L -> EQ | lambda

Q -> ,EQ | lambda

F -> function id H ( A ) { C }

H -> T | lambda

A -> T id K | lambda

K -> ,T id K | lambda

E -> UE’

E’ -> >UE’ | lambda

U -> VU’

U’ -> +VU’ | lambda

V -> G | entero | cadena | !G

G -> idV’ | ( E )

V’ -> -- | ( L ) | lambda

Y -> id | entero

W -> case entero : C W | break ; W | lambda

Z -> P | lambda

## Condiciones LL(1)

Solo vamos a realizar el análisis de producciones que poseen más de 2 reglas, ya que las producciones que poseen solo 1 regla siempre cumplen esta condición.

* Producción P:

First(BZ)= {let, if, switch, id, print, input, return}

First(FZ) = {function}

* Producción B:

First(let T id; ) = {let }

First(if ( E ) S) ={if}

First(switch ( Y ) { W }) ={switch}

First(S) = {id, print, return, input}

* Producción T:

First(int)={ int }

First(string) ={ string }

First(boolean) = { boolean }

* Producción S:

First(idS’)= {id}

First(print ( E ) ; )= {print}

First(input ( id ); )= {input}

First(return X;) = { return}

* Producción S’:

First(=E;)= { = }

First( ( L ) ) = { ( }

* Producción X:

First(E)= {id, (, entero, cadena, !}

Follow( X ) = { ; }

* Producción C:

First(BC)= {let, if, switch, id, print, input, return }

Follow( C ) = { }, case, break }

* Producción L:

First(EQ)= {id, (, entero, cadena, !}

Follow( L ) = { ) }

* Producción Q:

First(,EQ)= { , }

Follow( Q ) = { ) }

* Producción H:

First(T)= { int, string, boolean }

Follow( H ) = { ( }

* Producción A:

First(T id K ) ={ int, string, boolean }

Follow( A ) = { ) }

* Producción K:

First(,T id K )= { , }

Follow( K ) = { ) }

* Producción E’:

First(>UE’ ) ={ > }

Follow( E’ ) = { , , ; ,) }

* Producción U’:

First(+VU’ )= { + }

Follow( U’ ) = {>, , , ; , ) }

* Producción V:

First(G) ={id, (}

First(entero)= {entero}

First(cadena)= {cadena }

First(!G) = { ! }

* Producción G:

First(idV’)= { id }

First( ( E ) ) = { ( }

* Producción V’:

First( (L) )= { ( }

First(--)={--}

Follow( V’ ) = {+, >, , , ; , ) }

* Producción Y:

First(id ) ={ id}

First( entero ) = {entero }

* Producción W:

First(case entero : C W)= { case }

First(break ; W)= { break}

Follow(W) = { } }

* Producción Z:

First(P) ={ let, if, switch, id, print, input, return, function}

Follow(Z) = {$}

## Pseudo-Código

### Analizador Sintáctico

Function A\_Sint (){

sig\_tok=ALex();

P();

if (sig\_tok!="$") then error ()

}

Función Equipara

Function equipara (t){

if (sig\_tok==t)

then sig\_tok=ALex();

else error();

}

### Producciones

Function P() {

if(sig\_tok=={let, if, switch, id, print, input, return}) {

B();

Z();

}

else if(sig\_tok==function) {

F();

Z();

}

else {

Error("Token no esperado");

}

}

Function B() {

if(sig\_tok==let) {

equipara(let);

T();

equipara(id);

}

else if(sig\_tok==f) {

equipara(if);

equipara(();

E();

equipara());

S();

}

else if(sig\_tok==switch){

equpiara(switch);

equipara(();

Y();

equipara());

equipara({);

W();

equipara(});

}

else if(sig\_tok= {id, print, input, return}) {

S();

}

else {

Error("Token no esperado");

}

}

Function T() {

if(sig\_tok==int) {

equipara(int);

}

else if(sig\_tok==String) {

equipara(String);

}

else if(sig\_tok==Boolean) {

equipara(Boolean);

}

else {

Error("Token no esperado");

}

}

Function S()  {

        if(sig\_tok=="id") {

             print(10);

            equipara("id");

            S'();

        }

        else if(sig\_tok=="print") {

             print(11);

            equipara("print");

            equipara("(");

            E();

            equipara(")");

            equipara(";");

        }

        else if(sig\_tok=="input") {

             print(12);

            equipara("input");

            equipara("(");

            equipara("id");

            equipara(")");

            equipara(";");

        }

        else if(sig\_tok=="return") {

             print(13);

            equipara("return");

            X();

            equipara(";");

        }

        else {

            error();

        }

    }

    Function S'()  {

        if(sig\_tok=="=") {

             print(14);

            equipara("=");

            E();

            equipara(";");

        }

        else if(sig\_tok=="(") {

             print(15);

            equipara("(");

            L();

            equipara(")");

            equipara(";");

        }

        else {

            error();

        }

    }

    Function X()  {

        if(sig\_tok=="!" || sig\_tok=="id" || sig\_tok=="(" || sig\_tok=="entero"|| sig\_tok=="cadena" ) {

             print(16);

            E();

        }

        else if(sig\_tok==";") {

             print(17);

        }

        else {

            error();

        }

    }

    Function C()  {

        if(sig\_tok=="let" || sig\_tok=="if" || sig\_tok=="switch" || sig\_tok=="id" ||

                sig\_tok==("print") || sig\_tok==("input") || sig\_tok==("return")) {

             print(18);

            B();

            C();

        }

        else if(sig\_tok==("}") || sig\_tok==("case") || sig\_tok==("break")) {

             print(19);

        }

        else {

            error();

        }

    }

    Function L()  {

        if(sig\_tok=="id" || sig\_tok=="(" || sig\_tok=="entero" || sig\_tok=="cadena") {

             print(20);

            E();

            Q();

        }

        else if(sig\_tok==")") {

             print(21);

        }

        else {

            error();

        }

    }

    Function Q()  {

        if(sig\_tok==",") {

             print(22);

            equipara(",");

            E();

            Q();

        }

        else if(sig\_tok==(")")) {

             print(23);

        }

        else {

            error();

        }

    }

    Function F()  {

        if(sig\_tok=="function") {

             print(24);

            equipara("function");

            equipara("id");

            H();

            equipara("(");

            A();

            equipara(")");

            equipara("{");

            C();

            equipara("}");

        }

        else {

            error();

        }

    }

    Function H()  {

        if(sig\_tok=="int" || sig\_tok=="string" || sig\_tok=="boolean") {

             print(25);

            T();

        }

        else if(sig\_tok=="(") {

             print(26);

        }

        else {

            error();

        }

    }

    Function A()  {

        if(sig\_tok=="int" || sig\_tok=="string" || sig\_tok=="boolean") {

             print(27);

            T();

            equipara("id");

            K();

        }

        else if(sig\_tok==")") {

             print(28);

        }

        else {

            error();

        }

    }

    Function K()  {

        if(sig\_tok==",") {

             print(29);

            equipara(",");

            T();

            equipara("id");

            K();

        }

        else if(sig\_tok==")") {

             print(30);

        }

        else {

            error();

        }

    }

    Function E()  {

        if(sig\_tok=="id" || sig\_tok=="("|| sig\_tok=="entero" || sig\_tok=="cadena"|| sig\_tok=="!") {

             print(31);

            U();

            E'();

        }

        else {

            error();

        }

    }

    Function E'()  {

        if(sig\_tok==">") {

             print(32);

            equipara(">");

            U();

            E'();

        }

        else if(sig\_tok==")" || sig\_tok==";" || sig\_tok==",") {

             print(33);

        }

        else {

            error();

        }

    }

    Function U()  {

        if(sig\_tok=="id" || sig\_tok=="(" || sig\_tok=="entero" || sig\_tok=="cadena"|| sig\_tok=="!") {

             print(34);

            V();

            U'();

        }

        else {

            error();

        }

    }

    Function U'()  {

        if(sig\_tok=="+") {

             print(35);

            equipara("+");

            V();

            U'();

        }

        else if(sig\_tok==">" || sig\_tok==")" || sig\_tok==";" || sig\_tok==",") {

             print(36);

        }

        else {

            error();

        }

    }

V(){

if(sig\_tok=="id" || sig\_tok=="(") {

print(37);

G();

}

else if(sig\_tok==“entero")) {

print(38);

equipara("entero");

}

else if(sig\_tok=="cadena") {

print(39);

equipara("cadena");

}

else if(sig\_tok=="!") {

print(40);

equipara("!");

G();

}

else {

error();

}

}

G(){

if(sig\_tok.equals("id")) {

parse.add(41);

equipara("id");

V’();

}

else if(sig\_tok.equals("(")) {

parse.add(42);

equipara("(");

E();

equipara(")");

}

}

V’(){

if(sig\_tok== ("--")){

parse.add(43);

equipara("--");

}

else if(sig\_tok==("(")) {

parse.add(44);

equipara("(");

L();

equipara(")");

}

else if(sig\_tok== ("+") || sig\_tok== (">") || sig\_tok==(")") || sig\_tok==(";") || sig\_tok==(",")) {

parse.add(45);

}

else {

error();

}

}

    Function Y()  {

        if(sig\_tok=="id") {

             print(45);

            equipara("id");

        }

        else if(sig\_tok=="entero") {

             print(46);

            equipara("entero");

        }

        else {

            error();

        }

    }

    Function W()  {

        if(sig\_tok=="case") {

             print(47);

            equipara("case");

            equipara("entero");

            equipara(":");

            C();

            W();

        }

        else if(sig\_tok=="break") {

             print(48);

            equipara("break");

            equipara(";");

            W();

        }

        else if(sig\_tok==("}")) {

             print(49);

        }

        else {

            error();

        }

    }

Function Z()  {

if(sig\_tok=={let, if, switch, id, print, input, return, function}) {

             print(51);

            P();

        }

        else if(sig\_tok=="$") {

             print(52);

        }

        else {

            error();

        }

    }

### 

### Función Error

    Function error() {

            print("Error, el token no es el esperado");

            sig\_tok="";

    }

# Tabla de símbolos

Para esta primera entrega, existirá solamente una tabla de símbolos (ya que es solo el analizador léxico el que está trabajando). Así que lo único que sabremos de esta tabla es el lexema.

# Casos de prueba

## 

## Caso 1

let int a;

let int b;

let boolean bbb;

a = 3;

b=a;

let boolean c;

c = a > b;

if (c) bbb = bbb;

a = a + b;

print (a) ;

print(b);

### Tokens:

<let , >

<int , >

<id , 0>

<puntoComa , >

<let , >

<int , >

<id , 1>

<puntoComa , >

<let , >

<boolean , >

<id , 2>

<puntoComa , >

<id , 0>

<opAsignacion , >

<entero , 3>

<puntoComa , >

<id , 1>

<opAsignacion , >

<id , 0>

<puntoComa , >

<let , >

<boolean , >

<id , 3>

<puntoComa , >

<id , 3>

<opAsignacion , >

<id , 0>

<mayor , >

<id , 1>

<puntoComa , >

<if , >

<parentesisAbierto , >

<id , 3>

<parentesisCerrado , >

<id , 2>

<opAsignacion , >

<id , 2>

<puntoComa , >

<id , 0>

<opAsignacion , >

<id , 0>

<suma , >

<id , 1>

<puntoComa , >

<print , >

<parentesisAbierto , >

<id , 0>

<parentesisCerrado , >

<puntoComa , >

<print , >

<parentesisAbierto , >

<id , 1>

<parentesisCerrado , >

<puntoComa , >

### Parse:

Descendente 1 3 7 51 1 3 7 51 1 3 9 51 1 6 10 14 31 34 38 36 33 51 1 6 10 14 31 34 37 41 45 36 33 51 1 3 9 51 1 6 10 14 31 34 37 41 45 36 32 34 37 41 45 36 33 51 1 4 31 34 37 41 45 36 33 10 14 31 34 37 41 45 36 33 51 1 6 10 14 31 34 37 41 45 35 37 41 45 36 33 51 1 6 11 31 34 37 41 45 36 33 51 1 6 11 31 34 37 41 45 36 33 52

## Caso 2

let int n1;let int n2;

let boolean l1;let boolean l2;

let string cad;

input (n1);

l1 = l2;

if (! l2) cad = "hello";

n2 = n1 + 378;

print( 33 + n1 + n2);

function ff boolean(boolean ss){

varglobal = 8;

if (l1) l2 = ff (ss);

return ss;

}

if (ff(l2))

print (varglobal);

## Caso 3

let int x;

let boolean b;

let int z;

input(z);

print(z);

x=z;

print (z+1);

b=!b;if (b)

x =x + 6+ z+ 1+ (2+ y+ 7);

## Caso 4

let int x;

let int xx;

let string ss;

let boolean boolean\_1;

let boolean boolean\_2;

let int y;

print (4+5+77);

function f1 int(int f1, boolean b1)

{

print(ss);

x = xx+f1;

boolean\_1 = boolean\_2;

return (x);

}

function f2 boolean( int f2 , boolean b1 )

{

input (y);

print ((4+5+77+(088+f2)));

return (!b1);

}

x =x + 6+ z + 1+ (2 + y + 6);

print (f1 (x, f2 (3, boolean\_2)));

## Caso 5

let string cadena;

input(cadena);

let boolean logico1;

let boolean logico2;

let int int2;

int1 = 873;

int2 = 378;

if (! logico2) cadena = " hello";

int2=int2--;

function ff string(string sss)

{

global = 33;

logico1 = logico2;

if (logico1) sss = ff (cadena);

return sss;

}

function funcion string (string logico2)

{ let int var;

switch (int1){

case 0: logico1 = int1 > int2;break;

case 8888: print(0);

case 3333: logico2="";

}

return logico2;

}

print(((ff((funcion(cadena))))));

## Caso 1 (Error)

// Ejemplo #2

let string cadena;

input(cadena);

let boolean logico1;

let boolean logico2;

let int int2;

int1 = 873;

int2 = 378;

switch (int1){

case +: logico1 = int1 > int2;break;

case 8888: print(0);

case 3333: logico2="";

}

if (! logico2) cadena = " hello";

function ff string(string sss)

{

global = 33;

logico1 = logico2;

if (logico1) sss = ff (cadena);

return sss;

}

function funcion string (string logico2)

{ let int var;

return logico2;

}

print(((ff((funcion(cadena))))));

## Caso 2 (Error)

let int z;

let boolean boolean\_1;

let int x;

let string ss;

let int xx;

let boolean boolean\_2;

function f1 case(int f1, boolean b1){

input (z);

boolean\_1 = ! boolean\_2;

xx = f1+x;

print//Alerta

(ss);

return xx;

}

function f2 boolean( int f2 , boolean b1 ){

input (y);

print ((4+5+77+(088+f2)));

return (!!b1);

}

x =x + 6+ z + 1+ (2+ y + 6);

print (f1 (x, f2 (3, boolean\_2)));

## Caso 3 (Error)

let int b;

let int b;

let int a;

print ( "Introduce el primer operando" );

input (a);

print ("Introduce el segundo operando");input(b);

function operacion int(int num\_1, int num\_2){

return num\_1 + num\_2+77;

}

number = 0;

print(operacion(b,a));

function f2 boolean( int f2 , boolean b1 ){

input (y);

print ((4+5+77+(088+f2)));

return (!!b1);

}

z= x + 6+ z + 1+ (2 + 6);

let 30;

print (f5 (x, f2 (3, boolean\_2)));

## Caso 4 (Error)

let int66 = 32768;

let string cadena="Las constantes cadena van encerradas entre comillas dobles o entre comillas simples";

input(str);

let boolean bool1;

let boolean logico2;

let int int2;

int1 = 32;

int2 = 5;

if (! logico2 && bool1) cadena = " analizador";

function ff string(string sss)

{

global = 33/5;

logico1 = logico2;

logico1;

if (logico1 || logico2) sss = ff (cadena);

return sss;

}

function funcion string (string logico2)

{ let int var;

switch (int1){

case 0: logico1 = int1 < int2;break;

case 8888: print(0);

case 3333: logico2="";

}

return logico2;

}

## Caso 5 (Error)

let int z=4+2;

function f1 int(int z)

{

input (x);

let string = "cadena superior a los caracteres permitidos por JavaScript";

x = 5624642;

print(z)//Imprimimos z

}

let string ss;

let int xx;

let boolean boolean\_2;

function funcionF2 boolean(int f1, boolean b1)

{

input (z);

boolean\_1 = ! boolean\_2;

xx = f1+x;

print//Alerta

(ss);

return xx;

}

function funcionF2 string( int f2 , boolean b1 )

{

input (y);

print ((4+5+77+(088+f2)));

return (b1<b2);

}

x =x + 6458554+ z \* 1/(2+ aux\* 6);

print (x);