TRADUCTORES de lenguajes

Autores:

Daniel

Carlos Moro

Paula Copado González

**ÍNDICE**

1. DESCRIPCIÓN DEL DISEÑO FINAL
2. GCI
3. GENERADOR DE CÓDIGO FINAL
4. REGISTROS DE ACTIVACIÓN
5. CASOS DE PRUEBA
6. CONCLUSIÓN
7. **DESCRIPCIÓN DEL DISEÑO FINAL**

A la hora de implementar el código para nuestra práctica nosotros decidimos usar C, porque teníamos nociones de este lenguaje y además porque así podíamos usar algunas de las herramientas disponibles en la web de la asignatura para los analizadores sintáctico y semántico. A parte de las características comunes a todos los grupos nosotros teníamos que trabajar con la sentencia repetitiva for, con el operador especial post-auto-decremento (x--), el comentario de línea (//) y las cadenas con comillas simples (‘ ’).

El GCI recopila y serializa la información obtenida a través de los análisis léxico, sintáctico y semántico en cuartetos y, a partir de ellos, mediante un switch codificado con la operación a realizar traducimos estos cuartetos a instrucciones ensamblador que se puedan ejecutar.

1. **GCI**

**Gramática**

# Axioma

P'-> P

| P'.cod = P.cod

# Función

P -> FUN P1

| P.cod = FUN.cod || P1.cod

# Declaración

P -> DECL P1

| P.cod = DECL.cod || P1.cod

# Sentencias Compuestas

P -> SC P1

| P.cod = SC.cod || P1.cod

P -> λ

| P.cod = vacío

#And lógico

E -> E1 && T

| E.siguiente = nuevaEtiq()

| E.cod = E1.cod || gen("if", E1.lugar, "=", 1, "goto", E.siguiente) || gen (E.lugar, "=", "0") ||

| gen(E.siguiente, ":") || T.cod || gen("if", T.lugar, "=", 1, "goto", E.fin) || gen(E.lugar, "=", "0") ||

| gen(E.fin, ":") || gen(E.lugar, "=", "1")

# Expresiones y operaciones

E -> T

| E.lugar = T.lugar

| E.cod = T.cod

# Igual (operación)

T -> T1 == F

| T.cod = gen("if", T1.lugar, "=", F.lugar, "goto", T.siguiente) || gen(T.lugar, "=", "0") || gen(T.siguiente, ":") || gen(T.lugar, "=", "1")

T -> F

| T.lugar = F.lugar

| T.cod = F.cod

# Suma

F -> F1 + G

| F.lugar = nuevoTemp()

| F.cod = F1.cod || G.cod || gen(F.lugar, "=", F1.lugar, "+", G.lugar)

F -> G

| F.lugar = G.lugar

| F.cod = G.cod

G -> id

| G.lugar = buscaLugarTS(id.pos)

| G.cod = λ

# Postdecremento

G -> id--

| G.lugar = nuevoTemp()

| G.cod = gen(G.lugar, "=", (buscaLugarTS(id.pos)) || gen(buscaLugarTS(id.pos), "=", G.lugar,"-",1)

# Constantes

G -> true

| G.cod = gen(G.lugar, "=", 1)

G -> false

| G.cod = gen(G.lugar, "=", 0)

G -> cte\_ent

| G.lugar = nuevoTemp()

| G.cod = gen(G.lugar, "=", cte\_ent.valor)

G -> cadena

| G.lugar = nuevoTemp()

| G.cod = gen(G.lugar, "=", cadena.valor)

G -> (E)

| G.lugar = E.lugar

| G.cod = E.cod

# Llamada a función

G -> id(L)

| G.lugar = nuevoTemp()

| G.cod = L.codE || L.codP || gen(G. lugar:= "call", buscaEtiqTS(id.pos))

#Declaración de función

FUN -> function TIPO ID(PARAM) {Q}

| FUN.cod = gen(buscaEtiqTS(id.pos), ":") || Q.cod || gen("return")

Q -> DECL Q1

| Q.cod = DECL.cod || Q1.cod

Q -> SC Q1

| Q.cod = SC.cod || Q1.cod

Q -> λ

| Q.cod = vacío

# Declaración de variables

DECL -> var TIPOV id

| DECL.cod = vacío

# Sentencias compuestas

SC -> if (T) SS

| SS.siguiente = nuevaEtiq()

| SS.inicio = nuevaEtiq()

|

| SC.cod = T.cod || gen("if", T.lugar, "=", "0", "goto", SS.siguiente) || SS.cod || gen(SS.siguiente, ":")

SC -> FOR

| SC.cod = FOR.cod

# Sentencias simples

SC -> SS

| SC.lugar = SS.lugar

| SC.cod = SS.cod

SS -> print(E);

| SS.cod = E.cod || gen("print", E.lugar)

SS -> input(id)

| SS.lugar = nuevoTemp()

| SS.cod = gen("input",buscaLugarTS(id.pos))

SS -> return Z

| SS.cod = Z.cod || gen("return", Z.lugar)

SS -> AS

| SS.lugar = AS.lugar

| SS.cod = AS.cod

#Llamada a una función y parámetros

SS -> id(L)

| SS.cod = L.codE || L.codP || gen("call", buscaEtiqTS(id.pos))

L -> L1, E

| L.codE = L1.codE || E.cod

| L.codP = L1.codP || gen("param", E.lugar)

L -> E

| L.codE = E.cod

| L.codP = gen("param", E.lugar)

L -> λ

| L.codE = vacío

| L.codP = vacío

# Sentencias compuestas

W -> SC W1

| W.cod = SC.cod || W1.cod

W -> λ

| W.cod = vacío

# For

FOR -> for (PRE; R; POST) {W}

| FOR.inicio = nuevaEtiq()

| FOR.fin = nuevaEtiq()

| FOR.cod = PRE.cod || gen(FOR.inicio, ":") || E.cod || gen ("if", E.lugar, "=", "0", "goto", FOR.fin) || W.cod || POST.cod || gen("goto", FOR.inicio) || gen(FOR.fin, ":")

# Precondición

PRE -> id = E

| PRE.cod = E.cod || gen(buscaLugarTS(id.pos), "=", E.lugar)

PRE -> λ

| PRE.cod = vacío

# Postdecremento

POST -> id--

| POST.cod = gen(buscaLugarTS(id.pos),"=",buscaLugarTS(id.pos),"-",1)

POST -> λ

| POST.cod = vacío

# Asignación

AS -> id = E;

| AS.cod = E.cod || gen(buscaLugarTS(id.pos), "=", E.lugar)

1. **GCF**

DISEÑO

**R1:**

P'->P

| P'.cod = P.cod

**R2:**

P->B P

| P.cod = B.cod || P1.cod

**R3:**

P->F P

| P.cod = FUN.cod || P1.cod

**R4:**

P->

| P.cod = vacío

**R5:** #DECLARACION:

B->var T ID PuntoComa

| B.cod = vacío

**R6:**

B->if AbreParent E CierraParent S

**R7**:

B->do AbreCorchetes C CierraCorchetes while AbreParent E CierraParent PuntoComa

**R8:**

B->S

| B.lugar = S.lugar

| B.cod = S.cod

**R9:**

T->int

**R10:**

T->boolean

**R11:**

T->string

**R12:** # Asignación:

S->ID IGUAL E PuntoComa

| S.cod = E.cod || gen(buscaLugarTS(ID.pos), "=", E.lugar)

**R13:**

S->return X PuntoComa

**R14:**

S->print AbreParent E CierraParent PuntoComa

| S.cod = E.cod || gen("print", E.lugar)

**R15:**

S->input AbreParent ID CierraParent PuntoComa

**R16:**

S->ID AbreParent L CierraParent PuntoComa

**R17:**

X->E

**R18:**

X->

**R19:**

L->E Q

**R20:**

L->

**R21**:

Q->COMA E Q

**R22:**

Q->

**R23:**

F->function H ID AbreParent A CierraParent AbreCorchetes C CierraCorchetes

**R24:**

H->T

**R25:**

H->

**R25:**

C->B C

**R27:**

C->

**R28:**

A->T ID K

**R29:**

A->

**R30:**

K->COMA T ID K

**R31:**

K->

**R32:**

E->E AND R

**R33:**

E->R

| E.lugar = R.lugar

| E.cod = R.cod

| E.cadena = R.cadena

| E.digito = R.digito

**R34:**

R->R IGUALIGUAL U

| R.cod = gen("if", R1.lugar, "=", U.lugar, "goto", R.siguiente) || gen(R.lugar, "=", "0") || gen(R.siguiente, ":") || gen(R.lugar, "=", "1")

**R35:**

R->U

| R.lugar = U.lugar

| R.cod = U.cod

| R.cadena = U.cadena

| R.digito = U.digito

**R36:**

U->U Suma V

| U.lugar = nuevoTemp()

| U.cod = U1.cod || V.cod || gen(U.lugar, "=", U1.lugar, "+", V.lugar)

**R37:**

U->V

| U.TipoOperando = V.TipoOperando;

| U.Operando = V.Operando;

**R38:**

V->ID

| V.Operando = buscaLugarTS(ID.pos)

| V.TipoOperando = buscaTipoOperandoTS(ID.pos)

| V.cod = λ

**R39:**

V->digito

| V.lugar = Inmediato digito

**R40:**

V->true

| V.cod = gen(V.lugar, "=", 1)

**R41:**

V->false

| V.cod = gen(V.lugar, "=", 0)

**R42:**

V->cadena

| V.lugar = Inmediato cadena

**R43**:

V->ID AbreParent L CierraParent

**R44:**

V->AbreParent E CierraParent

**R45:**

V->ID MASMAS

|V = crearTemp

| gen(V = buscaLugarTS(ID)) || gen(ID = G.)

Hemos decidido implementar el GCO a través de los cuartetos que obteníamos gracias al GCI. Vamos a diferenciar cada uno de ellos a través de un código de operación que nos permita identificar qué tenemos que hacer en cada case del switch (**OP\_INPUT, OP\_PRINT, OP\_PLUS, OP\_AND, OP\_EQUALS, OP\_IF, OP\_GOTO, OP\_POST\_INC, OP\_ASIG, OP\_ETIQ**)

Nos hemos ayudado de unos atributos que hemos creado para poder especificar mejor cómo se tiene que realizar cada operación y con qué operandos, ya que al principio nos resultó difícil identificar cómo podíamos acceder a las posiciones de memoria donde se encontraba la información que necesitábamos para ejecutar el código. Nuestros cuartetos por lo tanto tienen el siguiente formato : “**Cuarteto(Operador operador, Atributo arg1, Atributo arg2, Atributo dest)”**

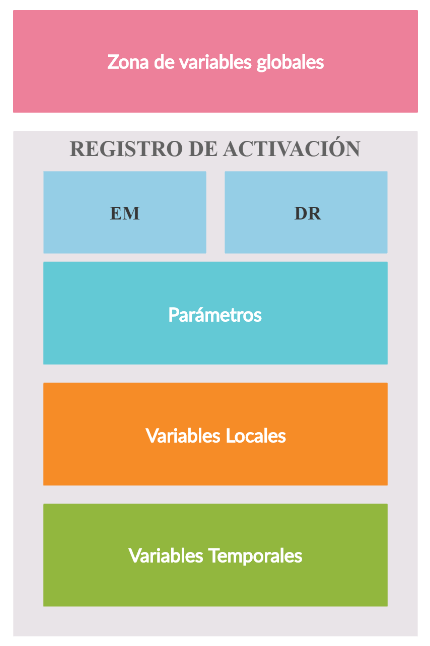
Dentro de cada case lo que hacemos es indicar las instrucciones de código ensamblador necesarias para poder ejecutar las operaciones con los operandos correspondientes.

1. **REGISTROS DE ACTIVACIÓN**

Para crear una estructura correcta de los registros de activación debemos tener en cuenta las distintas características del lenguaje.

No vamos a encontrarnos procedimientos anidados, por lo que no vamos a necesitar emplear el PA (Puntero de Acceso), y como vamos a asignar memoria mediante pila tampoco es necesario usar el PC (Puntero de Control). El EM y la DR sí se tendrán en cuenta, y se reservará espacio suficiente para almacenar los parámetros, las variables locales y las temporales en ese orden.

El resultado, dibujado de forma esquemática, sería el siguiente:



1. **CASOS DE PRUEBA**
2. //comment

function boolean f (int x){

String s= ‘hola mundo dfshxfg,.+-()’;

If(x ==0 && true)

print(s);

return x--;

}

1. var int v;

for(int i=6;i>0;i--){

input(v);

}

1. Input(id) ;
2. var int z;

function int suma (int x){

z = x + 3;

return z;

}

1. var int i = 75; // i es un numero entero

print (i);

var string p;

p= 'pepito';

print (p);

1. **CONCLUSIÓN**

A pesar de que esta práctica ha resultado ser todo un desafío y nos ha llevado bastante esfuerzo, ha sido una oportunidad para ver cómo de capaces somos de hacer proyectos nuevos y desarrollar nuestra capacidad de esfuerzo y trabajo en equipo.

Las principales dificultades que encontramos durante la realización de la práctica han estado sobre todo relacionadas con la organización de la memoria y cómo podíamos extraer la información necesaria para realizar las operaciones. También ha supuesto un reto el asignar las etiquetas en las operaciones de ensamblador y definir de forma correcta los cuartetos.

Por último creemos que a pesar de todo hemos aprendido mucho haciendo este proyecto y ha sido muy interesante poder aplicar todo lo aprendido en clase en un caso real, y crear nuestro propio Traductor de Lenguajes.