Esquema de traducción

Concha Vázquez Miguel
mconcha@ciencias.unam.mx

Flores Martínez Andrés
andresfm97@ciencias.unam.mx

Gladín García Ángel Iván
angelgladin@ciencias.unam.mx

Sánchez Pérez Pedro Juan Salvador
pedro_merolito@ciencias.unam.mx

Vázquez Salcedo Eduardo Eder
eder.vs@ciencias.unam.mx

14 de diciembre de 2018

prog → decls funcs

prog → decls funcs

decls → tipo lista ; decls

decls → tipo {current_dim = decls.bytes; current_dim_arr = current_dim; current_arr_type = current_type;} lista ; decls1 {decls.cantidad = lista.cantidad; decls.cantidad += decls1.cantidad;}

decls → ϵ

decls → ϵ { decls.cantidad = 0;}

tipo → int

tipo → int {tipo.type = 0; tipo.bytes = 4; }

tipo → float

tipo → float {tipo.type = 1; tipo.bytes = 4; }

tipo → double

tipo → double {tipo.type = 2; tipo.bytes = 8; }

tipo → char

tipo → char {tipo.type = 3; tipo.bytes = 1; }

tipo → void

tipo → void {tipo.type =4; tipo.bytes = 1; }

tipo → struct { decls }

tipo → struct { {

Table.symbolTable = new SymbolTable();
Table.typeTable = new TypeTable();

TableStack.push(Table) } decls } {

sacada = pop(TableStack)
renglon = newTypeTableRow();
renglon.type = 6;
renglon.tam = decls.cantidad;
renglon.base.renglon = -2;
renglon.base.smt = sacada;
insert_type_table(TableStack.typeTable, renglon);
struct.type = TableStack.typeTable.count-1; struct.bytes = decls.cantidad;

}

lista → lista1, ID arreglo

lista → lista1, ID arreglo {

s = new SymbolTableRow();
s.id = ID.yylval;
s.type = current_arr_type;
s.dir = dir;
dir += current_dim_arr;
insert(TableStack.symbolTable, s)
lista.cantidad = current_dim_arr;
current_arr_type = current_type;
current_dim_arr = current_dim;

}

lista → ID arreglo

lista → ID arreglo {

s = new SymbolTableRow();
s.id = ID.yylval;
s.type = current_arr_type;
s.dir = dir;
dir += current_dim_arr;
insert(TableStack.symbolTable, s)
lista.cantidad = current_dim_arr;
current_arr_type = current_type;
current_dim_arr = current_dim;

}

numero → signo INT

numero → signo INT { numero.type = INT.yylval.type; numero.val = signo; numero.val += INT.yylval.val; }

numero → signo DOUBLE

numero → signo DOUBLE { numero.type = DOUBLE.yylval.type; numero.val = signo; numero.val += DOUBLE.yylval.val; }

numero → signo FLOAT

numero → signo FLOAT { numero.type = FLOAT.yylval.type; numero.val = signo; numero.val += FLOAT.yylval.val; }

signo → +

signo → + { signo = " "; }

signo → -

signo → - { signo = "-"; }

signo → ϵ

signo → ϵ { signo = " "; }

arreglo → [numero] arreglo

arreglo → [numero] arreglo1 {

```
arreglo.tam = arreglo1.tam + 1;
current_dim_arr = current_dim_arr * numero.val;
arreglo.dims = arreglo1.dims;
arreglo.dims[arreglo.tam-1] = toInt(numero.val);
renglon = new TypeTableRow();
renglon.type = 5;
renglon.tam = numero.val;
renglon.base.renglon = current_arr_type;
renglon.base.smt = NIL;
insert_type_table(TableStack.typeTable, renglon);
current_arr_type = TableStack.typeTable.count-1;
```

}

arreglo → ϵ

arreglo → ϵ { arreglo.tam = 0; }

funcs → func tipo ID (args) decls sents funcs

funcs → funcs tipo id {

Table1.tt = newTypeTable();
Table1.st = newSymbolTable();
mete(TableStack, Table1);
current_function_type = tipo.type;

} (args) {

cuadrupla c;
char lab[32];
funcs.code = id + ":" SymbolTableRow.id = id;
SymbolTableRow.type = tipo.type;
SymbolTableRow.dir = Table.TypeTable.count;
SymbolTableRow.var = 1;
SymbolTableRow.args = args.lista_args;
SymbolTableRow.num_args = args.num;
insert(Table.SymbolTable, SymbolTableRow);

} { decls sents } {

funcs.code += get_first(sents) + ":" backpatch(sents, newLabel());
sacada = saca(TableStack);
TypeTableRow1.type = 7;
TypeTableRow1.tam = decls.cantidad;
TypeTableRow1.base.renglon = -2;
TypeTableRow1.base.smt = sacada;
insert_type_table(StackTable.tabla.TypeTable, TypeTableRow1);
funcs.code += newLabel() + ":"

}

funcs → ϵ

funcs → ϵ

args → lista_args

args → lista_args {

args.num = lista_args.num;
args.lista_args = lista_args.lista_args;

}

lista_args → lista_args1, tipo ID parte_arr

lista_args → lista_args1, tipo ID parte_arr {

TypeTableRow.type = tipo.type;
TypeTableRow.tam = tipo.type;
TypeTableRow.base.renglon = tipo.type;
insert_type_table(TableStack.tabla.TypeTable, TypeTableRow);
SymbolTableRow.id = ID.yylval;
SymbolTableRow.type = tipo.type;
SymbolTableRow.dir = dir;
dir += TypeTableRow.tam;
insert(TableStack.tabla.SymbolTable, SymbolTableRow);
lista_args.num = 1 + lista_args1.num;
lista_args.lista_args = lista_args1.lista_args;
lista_args.lista_args[lista_args.num-1][0] = tipo.type;
lista_args.lista_args[lista_args.num-1][1] = parte_arr;

}

lista_args → tipo ID parte_arr

lista_args → tipo ID parte_arr {

lista_args.num = 1;
lista_args.lista_args[0][0] = tipo.type;
lista_args.lista_args[0][1] = parte_arr;
TypeTableRow.type = tipo.type;
TypeTableRow.tam = tipo.type;
TypeTableRow.base.renglon = tipo.type;
insert_type_table(StackTable.tabla.TypeTable, TypeTableRow);
SymbolTableRow.id = ID.yylval;
SymbolTableRow.type = tipo.type;
SymbolTableRow.dir = dir;
dir += renglon.tam;
insert(StackTable-.tabla-.SymbolTable, SymbolTableRow);

}

parte_arr → [] parte_arr1

parte_arr → [] parte_arr1 { parte_arr = parte_arr1 + 1; }

parte_arr → ϵ

parte_arr → ϵ { parte_arr = 0; }

`sents → sents1 sent`

`sents` → `sents1` {

sents.code = get_first(sents1) + ":"; } `sent` {

sents = sent;
backpatch(sents, newLabel(););

}

`sents → sents1 sent`

`sents` → `sent` {

backpatch(sent, newLabel(););

}

`sent → if ( cond ) sent1 sentp`

`sent` → `if (cond)` {

sent.code = get_first(cond.trues) + ":";

} `sent` { cuadrupla c;
sent.code += "goto " + get_first(sent1);
push_label(&lfalses, get_first(cond.falses));
} `sentp` {

if(sentp.ifelse == false) {
label = newLabel();
sent = merge(cond.falses, sentp.siguientes);
backpatch(cond.falses, label);
} else {
label2 = newLabel();
sent = merge(sent1, sentp.siguientes);
backpatch(cond.trues, label2, codigo_intermedio);
backpatch(cond.falses, label, codigo_intermedio);
}

}

`sent → while ( cond ) sent1`

`sent → while (cond) {`

breakeablecitos += 1;
siguiente_breakable_pila[siguiente_count] = < *siguientes* >.label[0];
siguiente_count++;

`} sent1 {`

label = newIndex();
label2 = newIndex();
sent = cond.falses;
sent = “id ” + cond.code + “ goto ” + label + “;”;
backpatch(cond.trues, label);
backpatch(cond.falses, label2);
breakeablecitos -= 1;
siguiente_breakable_pila[siguiente_count] = < *siguientes* >.label[0];
siguiente_count–;

}

`sent → do sent while ( cond );`

`sent1 → do {`

breakeablecitos += 1;
siguiente_breakable_pila[siguiente_count] = < *siguientes* >.label[0];
siguiente_count++;

`} while (cond); {`

char label[32], label2[32], temp[32];
label = newIndex();
label2 = newIndex();
temp = newTemp();
sent.code = “iff ” + code.code + “ goto ” + label + “;”;
backpatch(cond.trues, label);
backpatch(cond.falses, label2);
breakeablecitos -= 1;
siguiente_breakable_pila[siguiente_count] = < *siguientes* >.label[0];
siguiente_count–;

}

sent → for (assign; cond; assign) sent1

sent → for (assign1; cond; assign2) {

breakeablecitos += 1;
siguiente_breakable_pila[siguiente_count] = < *siguientes* >.label[0];
siguiente_count++;

} sent1 {

meter_assign(assign1.arr_codigo, assign1.count_codigo);
label = newIndex();
label2 = newIndex();
sent = cond.falses;
sent.code = “iff ” + code.code + “ goto ” + label + “;”;
backpatch(&cond.trues, label, &codigo_intermedio);
backpatch(&cond.falses, label2, &codigo_intermedio);
meter_assign(assign2.arr_codigo, assign2.count_codigo);
breakeablecitos -= 1;
siguiente_count–;

}

sent → assign;

sent → assign; {meter_assign(assign.arr_codigo, assign.count_codigo);}

sent → return exp;

sent → return exp; {

sent.code = “goto ” + label1;

}

sent → return exp;

sent → return; {

sent.code = “goto ” + label1;

}

sent → {sents};

sent → {sents} {

label = newLabel();
backpatch(&sents, label);

}

sent → switch (exp) {casos};

sent → switch (exp) {

breakeablecitos++;
siguiente_breakable_pila[siguiente_count] = < *siguientes* >.label[0];
siguiente_count++;
pila_switch[count_switch] = exp.dir;
count_switch++;

} {casos} {

count_switch–;
breakeablecitos–;
siguiente_count–;

}

sent → break;

sent → break; {

label1 = newLabel();
sent.code = "goto " + label1;

}

sent → print exp;

sent → print exp; {

i = newIndex(); sent = create_list(i); }

assign → parte_izq = exp

assign → parte_izq = exp {

i = newIndex();
assign = create_list(i);
e = asignacion(parte_izq.id1, parte_izq.id2, exp, parte_izq.type);
assign.count_codigo = e.count_codigo;
assign.arr_codigo = e.arr_codigo;

}

sentp → ϵ

assign → ϵ { sentp.ifelse = false; }

sentp → else sent

assign → else {

assign.code = pop_label(&lfalses) + ":";

} sent {

sentp.ifelse = true;
sentp.siguientes = sent;

}

casos → case numero sent casos1

casos → case numero {

i = newIndex(); i2 = newIndex(); temp = newTemp(); current_label = i2;
< *booleanos* >.trues = create_list(i);
< *booleanos* >.falses = create_list(i2);
c.op = EQUALS;
char tope_dir[32];
casos.code = temp + " = " + tope_dir + " = " + numero.val + ";"; tope_dir = pila_switch[count_switch-1]; c.op1 = tope_dir; c.op2 = numero.val; c.res = temp;
c1.op = IFF; c1.op1 = temp; c1.op2 = "GOTO"; c1.res = i; c2.op = GOTO; c2.op1 = ""; c2.op2 = ""; c2.res = i2;
insert_cuad(&codigo_intermedio, c);
insert_cuad(&codigo_intermedio, c1);
insert_cuad(&codigo_intermedio, c2);
cuadrupla c3;
c3.op = LABEL; c3.op1 = ""; c3.op2 = ""; c3.res = i;
insert_cuad(&codigo_intermedio, c3);

} sent {

cuadrupla c, c1;
c.op = LABEL; c.op1 = ""; c.op2 = ""; c.res = current_label;
push_label(&lfalses, get_first(&< *booleanos* >.falses));
insert_cuad(&codigo_intermedio, c);

} casos1

casos → default sent

casos → default sent {}

casos → ϵ

casos → ϵ {}

parte_izq → ID

parte_izq → id { parte_izq.id1 = ID.yylval; parte_izq.id2 = ; parte_izq.type = -1; }

parte_izq → var_arr

parte_izq → var_arr { parte_izq.id1 = var_arr.representacion; parte_izq.type = var_arr.type; }

parte_izq → ID1.ID2

parte_izq → id1.id2 { parte_izq.id1 = ID1.yylval; parte_izq.id2 = ID2.yylval; parte_izq.type = -1; }

var_arr → ID[exp]

var_arr → id[exp] {

```
var_arr.representacion = ID.yylval + "[" + exp.dir + "]";
struct nodo* it = stack_mastertabs;
int encontrado = 0;
while(it != NULL) {
int x = search(it->tabla->st, id);
if(x != -1) {
encontrado = 1;
int type_row = it->tabla->st->symbols[x].type;
var_arr.type = it->tabla->tt->trs[type_row].base.renglon;
var_arr.tipo_basico = var_arr.type;
var_arr.tamanios[0] = it->tabla->tt->trs[var_arr.tipo_basico+1].tam;
int mydims = 1;
while(var_arr.tipo_basico >4){
var_arr.tamanios[mydims] = it->tabla->tt->trs[var_arr.tipo_basico].tam;
mydims++;
var_arr.tipo_basico = it->tabla->tt->trs[var_arr.tipo_basico].base.renglon;
}
var_arr.indice_tamanios++;
var_arr.dims = mydims - 1;
break;
}
it = it->siguiente;
}
var_arr.tt = it->tabla->tt;

}
```

var_arr → var_arr1 [exp]

var_arr → var_arr1 [exp] {

```
var_arr.tipo_basico = var_arr1.tipo_basico;
var_arr.dims = var_arr1.dims - 1;
var_arr.representacion = id + "[" + exp.dir + "]";
int row_hijo = var_arr1.type;
var_arr.type = (*var_arr1.tt).trs[row_hijo].base.renglon;
var_arr.indice_tamanios++;
var_arr.tt = var_arr1.tt;

}
```

exp → exp1 + exp2

exp → exp1 + exp2 { exp = suma(exp1, exp2) }

exp → exp1 - exp2

exp → exp1 - exp2 { exp = resta(exp1, exp2) }

exp → exp1 * exp2

exp → exp1 * exp2 { exp = multiplicacion(exp1, exp2) }

exp → exp1 / exp2

exp → exp1 / exp2 { exp = division(exp1, exp2) }

exp → exp1 mod exp2

exp → exp1 mod exp2 { exp = modulo(exp1, exp2) }

exp → var_arr

exp → var_arr {

exp = envolver_varr(var_arr);
exp.tipo_basico = var_arr.tipo_basico;
exp.dims = var_arr

}

exp → exp1 / exp2

exp → exp1 / exp2 { exp = division(exp1, exp2) }

exp → ID

exp → ID {exp = identificador(ID.yylval);}

exp → CADENA

exp → CADENA {exp = envolver_cadena(CADENA.yylval);}

exp → numero

exp → numero {exp = get_numero(numero);}

exp → CARACTER

exp → CARACTER {exp = envolver_caracter(CARACTER.yylval);}

exp → ID (params)

exp → ID (params) {

verifica_call(ID.yylval, params.lista_tipos, params.count);
char temp[32];
temp = newTemp();
cuadrupla c;
c.op = CALL; c.op1 = ID.yylval;
sprintf(c.op2, ç.res = temp;
insert_cuad(&codigo_intermedio, c);
exp.dir = temp;

}

params → lista_param

params → lista_param {

params.p = lista_param.p;
params.lista_tipos = lista_param.lista_tipos;
params.count = lista_param.count;

}

params → ϵ

params → ϵ {

params.p = 0;
params.count = 0;

}

lista_param → lista_param1, exp

lista_param → lista_param1, exp {

cuadrupla c;
c.op = PARAM; c.op1 = “”; c.op2 = “”; c.res = exp.dir;
insert_cuad(&codigo_intermedio, c);
lista_param.p = lista_param1.p + 1;
lista_param.lista_tipos[lista_param1.count][0] = exp.tipo_basico;
lista_param.lista_tipos[lista_param1.count][1] = exp.dims;
lista_param.count = lista_param1.count + 1;

}

`lista_param → exp`

`lista_param → exp {`

cuadrupla c;
c.op = PARAM; c.op1 = “”; c.op2 = “”; c.res = exp.dir;
insert_cuad(&codigo_intermedio, c);
lista_param.p = 1;
lista_param.lista_tipos[0][0] = exp.tipo_basico;
lista_param.lista_tipos[0][1] = exp.dims;
lista_param.count = 1;

}

`cond → cond1 || cond2`

`cond → cond1 || {`

cuadrupla c;
c.op = LABEL; c.op1 = “”; c.op2 = “”; c.res = get_first(&cond1.falses);
insert_cuad(&codigo_intermedio, c);

}cond2 {

char label[32];
label = newLabel();
backpatch(&cond1.falses, label, &codigo_intermedio);
cond.trues = merge(&cond1.trues, &cond2.trues);
cond.falses = cond2.falses;

}

`cond → cond1 && cond2`

`cond → cond1 && {`

cuadrupla c;
c.op = LABEL; c.op1 = “”; c.op2 = “”; c.res = get_first(&cond1.trues);
insert_cuad(&codigo_intermedio, c);

}cond2 {

char label[32];
label = newLabel();
cond.falses = merge(&cond1.falses, &cond2.falses);
cond.falses = cond2.trues;
backpatch(&cond1.trues, label, &codigo_intermedio);

}

`cond → ! cond1`

`cond → ! cond1` {cond.falses = cond1.trues; cond.trues = cond1.falses;}

cond → (cond1)

cond → (cond1) {cond.trues = cond1.trues; cond.falses = cond1.falses}

cond → exp1 rel exp2

cond → exp1 rel exp2 {

char i[32], i2[32], temp[32];
i = newIndex(); i2 = newIndex(); temp = newTemp();
cond.trues = create_list(i);
cond.falses = create_list(i2);
cuadrupla c, c1, c2;
c.op = rel ; c.op1 = exp1.dir; c.op2 = exp2.dir; c.res = temp;
c1.op = IFF; c1.op1 = temp; c1.op2 = "GOTO"; c1.res = i;
c2.op = GOTO; c2.op1 = “”; c2.op2 = “”; c2.res = i2;
insert_cuad(&codigo_intermedio, c);
insert_cuad(&codigo_intermedio, c1);
insert_cuad(&codigo_intermedio, c2);

}

cond → true

cond → true {

i = newIndex();
cond.trues = create_list(i);
}

cond → false

cond → false {

i = newIndex();
cond.falses = create_list(i);
}

rel → <

rel → < {rel.tipo = LESS_THAN; }

rel → >

rel → > {rel.tipo = GREATER_THAN; }

rel → <=

rel → <= {rel.tipo = LESS_OR_EQUAL_THAN; }

rel → >=

rel → >= {rel.tipo = GREATER_OR_EQUAL_THAN; }

`rel → !=`

`rel` → ! = {rel.tipo = NOT_EQUALS; }

`rel → ==`

`rel` → == {rel.tipo = EQUALS; }