GENERADOR DE CÓDIGO INTERMEDIO

Juan Alberto Cañero Tamayo

GENERADOR DE CÓDIGO INTERMEDIO

Realizado por: Juan Alberto Cañero Tamayo

A continuación se expone un generador de código intermedio. Para la total comprensión de las distintas tareas a realizar, he decidido implementar todo el código desde cero, tomando como base los apuntes de la asignatura.

Para ello empezamos implementando el sistema de tipos. Una vez descrito el sistema de tipos ya podemos desarrollar la tabla de símbolos, para más tarde enumerar los tokens y por último describir la gramática en la que se apoyará YACC.

EL SISTEMA DE TIPOS

Entre las distintas opciones de diseño, nos hemos decantado por emplear la representación de tipos codificada, dado que para la dimensión de la práctica es más que suficiente.

tipos.h

```
#define tipoerror   0
#define tipochar    1
#define tipoint     2
#define tiporeal    3

#define tipopuntero 1
#define tipoarray   2
#define tipofuncion 3
```

El prototipo de las funciones empleadas son:

```
int componertipo( int constructor , int tipo);
// Dado un tipo base, devuelve la composición de dicho tipo con un constructor
int tiposimple( int tipo );
// Devuelve 1 si es un tipo simple
int tipobase( int tipo );
// Devuelve el tipo base de un tipo compuesto
int comparartipos( int t1 , int t2 );
// Devuelve 1 se los tipos son iguales
int tamtipo( int t );
// Devuelve el tamaño de un tipo dado
```

tipos.c

```
#include "tipos.h"

int componertipo( int constructor , int tipo){

       return(4*tipo+constructor);
} /*componertipo*/

int tipobase( int tipo ){

       return( tipo/4 );
} /*tipobase*/

int tiposimple( int tipo ){

       return( tipo < 4 );
} /*tiposimple*/

int comparartipos( int t1 , int t2 ){

       return( t1 == t2 );
} /*comparartipos*/
```

```
int tamtipo( int t ){
        int sz;

        switch (t) {
                case tipoint   : sz=4 ; break;
                case tiporeal  : sz=8 ; break;
                case tipochar  : sz=1 ; break;
                case tipoerror : sz=0 ; break;
                default : sz=4;
        } /*switch*/
        return( sz );
} /*tamtipo*/
```

LA TABLA DE SÍMBOLOS

La opción de diseño aquí elegida ha sido una tabla hash. Para los lexemas en almacenados en la tabla hash se ha optado por una implementación interna. Cada nodo de la tabla hash debe tener un indicador de bloque para saber a que bloque pertenece cada lexema, adicionalmente le hemos añadido un campo para conocer la dirección de memoria a la que cual va a hacer referencia (a título informativo).

El tratamiento de los desbordamientos se ha tratado con ayuda de una lista enlazada.

tablasim.h

```
typedef struct node {
        char nombre[16];              // Nombre del símbolo
        int  direccion;               // Dirección relativa en memoria
        int  tam;                     // Tamaño reservado
        int  tipo;                    // Tipo de dato
        int  bloque;                  // Número de bloque al que pertenece
        struct node *sig;             // Puntero a siguiente
} NODO;

void inicializar_tabla();

void insertar( NODO n );

int existe( char *clave );

NODO *buscar( char *clave );          // Nótese la diferencia entre buscar y existe

void nuevobloque();

void finbloque();
```

tablasim.c

```
#define hash_size 5                   // Tamaño de la tabla hash
#include "tablasim.h"

NODO *tabla[hash_size];
int NumBloque;


/* -- Funciones referentes a listas ---------------------------------------- */

NODO *nueva_lista(){

        return( (NODO *) 0 );
} /*nueva_lista*/


NODO *insertar_lista( NODO *lista , NODO item ){
/* Creo un nuevo nodo con malloc y apunto a la lista */
        NODO *nuevo;
```

```
        nuevo = (NODO *) malloc( sizeof(NODO) );
        strcpy( (*nuevo).nombre , item.nombre );
        (*nuevo).direccion = item.direccion;
        (*nuevo).tam       = item.tam;
        (*nuevo).tipo      = item.tipo;
        (*nuevo).bloque    = item.bloque;
        (*nuevo).sig       = lista;

        return( nuevo );
} /*insertar_lista*/


int buscar_lista( NODO *lista , char *clave ){
        int encontrado=0;

        while( (lista != (NODO *) 0) && !encontrado ) {
                encontrado=!strcmp( (*lista).nombre , clave );
                lista = (*lista).sig;
        } /*while*/
        return( encontrado );
} /*buscar_lista*/


NODO *recuperar_nodo_lista( NODO *lista , char *clave ){
        int encontrado=0;
        NODO *recuperado;

        recuperado = (NODO *) 0;
        while( (lista != (NODO *) 0) && !encontrado ) {
                if ( !strcmp( (*lista).nombre , clave ) ){
                        recuperado=lista;
                        encontrado=1;
                } /*if*/
                lista = (*lista).sig;
        } /*while*/
        return( recuperado );
} /*recuperar_nodo_lista*/


void borrar_lista( NODO *lista ){
        NODO *borrable;

        while( lista != (NODO *) 0 ) {
                borrable = lista;
                lista = (*lista).sig;
                free( borrable );
        } /*while*/
} /*borrar_lista*/


NODO *borrar_coincidentes( NODO *lista , int bloque ){
        NODO *act,*ant,*borrable;
        int pos,i;
        int res;

        act = ant = lista;
        while ( (act != (NODO *) 0 ) ) {
                if ( (*act).bloque == bloque ) {
                        if ( act == ant ) {
                                lista = (*act).sig;
                                ant=(*act).sig;
                        } /*if*/

                        else
                                (*ant).sig = (*act).sig;
                        borrable=act;
                        act = (*act).sig;
                        free(borrable);
                } /*if*/
                else {
                        ant = act;
                        act = (*act).sig;
                } /*else*/
        } /*while*/

        return( lista );
} /* borrar_coincidentes */
```

```
/* -- Funciones referentes a la tabla de simbolos -------------------------- */


int numerohash( char *cadena ){
	long valor=0;
	int f;

	for( f = 0 ; f <= strlen(cadena) ; f++ )
		valor = valor + (long)(cadena[f]+f);

	return( (int) ( valor % (long) hash_size ) );
} /* numerohash */


void inicializar_tabla(){
	int i;

	for( i = 0 ; i < hash_size ; i++ )
		tabla[i]=nueva_lista();
	NumBloque=0;
} /*inicializar_tabla*/


void insertar( NODO n ){
	NODO *lista;
	int hashval;

	hashval=numerohash(n.nombre);
	lista=tabla[ hashval ];
	n.bloque = NumBloque;
	lista=insertar_lista( lista , n );
      tabla[ hashval ]=lista;
} /*insertar*/


int existe( char *clave ){
	NODO *lista;

	lista=tabla[ numerohash(clave) ];
	return( buscar_lista(lista,clave) );
} /*existe*/


NODO *buscar( char *clave ){
	NODO *lista;

	lista=tabla[ numerohash(clave) ];
	return( recuperar_nodo_lista(lista,clave) );
} /*buscar*/


void nuevobloque(){

	NumBloque++;
} /*nuevobloque*/



void finbloque(){
	int i;

	for( i=0 ; i<hash_size ; i++)
		tabla[i] = borrar_coincidentes( tabla[i] , NumBloque );
	NumBloque--;
} /*finbloque*/
```

Definición de los lexemas (Tokens)

La primera parte del análisis lo desarrollaremos con ayuda de LEX, un analizador lexicográficos muy vinculado con el YACC. En él describimos también las palabras reservadas: for, if, break, int, ...etc

```
%{
#include "y.tab.h"
%}
```

```
num                     0|[1-9][0-9]*
alpha                   [A-Za-z]
ident                   {alpha}[_A-Za-z0-9]*

%%

"//".*                  ;

repeat                  return REPEAT;
for                     return FOR;
switch                  return SWITCH;
case                    return CASE;
default                 return DEFAULT;
break                   return BREAK;
until                   return UNTIL;
if                      return IF;
else                    return ELSE;
elsif                   return ELSIF;
int                     return INT;
real                    return REAL;
char                    return CHAR;
true                    return TRUE;
false                   return FALSE;
do                      return DO;
while                   return WHILE;
step                    return STEP;
to                      return TO;
downto                  return DOWNTO;

{num}                   {
                        sprintf(yylval.cadena,"%s",yytext);
                        return ENTERO;
                        }
{num}\.[0-9]*           {
                        sprintf(yylval.cadena,"%s",yytext);
                        return NREAL;
                        }
{ident}                 {
                        sprintf(yylval.cadena,"%s",yytext);
                        return IDENT;
                        }

\{                      return '{';
\}                      return '}';
\[                      return '[';
\]                      return ']';
\(                      return '(';
\)                      return ')';

=                       return '=';
\+                      return '+';
\-                      return '-';
\*                      return '*';
\/                      return '/';
%                       return '%';
"++"                    return MASMAS;
"--"                    return MENMEN;
"+="                    return MASIG;
"-="                    return MENIG;
\?                      return '?';

&&                      return AND;
"||"                    return OR;
!                       return NOT;
\>                      return MAYOR;
\<                      return MENOR;
"<="                    return MENORIG;
">="                    return MAYORIG;
"=="                    return IGIG;

:                       return ':';
;                       return ';';
,                       return ',';


" "|\t|\n               ;

.                       printf("<Atención: carácter no reconocido e ignorado:\"%c\" >\n",yytext[0]);

%%
```

BIBLIOTECA PARA IMPRIMIR LAS INSTRUCCIONES

Antes de mostrar la gramática, hemos de reparar en que necesitamos una biblioteca para la generación del código correspondiente al código de 3 direcciones, el juego completo de instrucciones que disponemos van a ser: A y B representan variables y/o constantes, L representa una etiqueta (en la siguiente fase... Una dirección de memoria)

```
variable = A

variable = variable[desplazamiento]

variable[desplazamiento] = variable

variable = A + B

variable = A - B

variable = A * B

variable = A / B

IF A > B GOTO L

GOTO L

LABEL L:
```

Además se incluyen funciones para la generación de nombres de variables temporales y etiquetas.

gc.h

```
#define C3D_ASIG 1010
#define C3D_AARR 1012
#define C3D_ARRA 1014
#define C3D_SUMA 1020
#define C3D_RSTA 1025
#define C3D_PROD 1040
#define C3D_DIVI 1045
#define C3D_IFMA 1050
#define C3D_IFIG 1055
#define C3D_GOTO 1060
#define C3D_LABL 1070

void gc( int instruccion , char *argumento1 , char *argumento2 , char *resultado );
char *nuevaTemporal();
char *nuevaEtiqueta();
```

gc.c

```
#include <stdio.h>
#include "gc.h"

extern FILE *yyout;

void gc( int instruccion , char *argumento1 , char *argumento2 , char *resultado ){

        switch( instruccion ) {
                case C3D_ASIG : { fprintf( yyout , "%s = %s" , resultado , argumento1 );
                                      break; }
                case C3D_AARR : { fprintf( yyout , "%s = %s[%s]" , resultado , argumento1 ,
                                      argumento2 );
                                      break; }
                case C3D_ARRA : { fprintf( yyout , "%s[%s] = %s" , resultado , argumento1 ,
                                      argumento2 );
                                      break; }
                case C3D_SUMA : { fprintf( yyout , "%s = %s + %s" , resultado , argumento1 ,
                                      argumento2 );
                                      break; }
                case C3D_RSTA : { fprintf( yyout , "%s = %s - %s" , resultado , argumento1 ,
                                      argumento2 );
                                      break; }
                case C3D_PROD : { fprintf( yyout , "%s = %s * %s" , resultado , argumento1 ,
                                      argumento2 );
                                      break; }
```

```
            case C3D_DIVI : { fprintf( yyout , "%s = %s / %s" , resultado , argumento1 ,
                                  argumento2 );
                                  break; }
            case C3D_IFMA : { fprintf( yyout , "IF %s > %s GOTO %s" , argumento1 , argumento2 ,
                                  resultado );
                                  break; }
            case C3D_IFIG : { fprintf( yyout , "IF %s = %s GOTO %s" , argumento1 , argumento2 ,
            resultado );
                                  break; }
            case C3D_GOTO : { fprintf( yyout , "GOTO %s" , resultado );
                                  break; }
            case C3D_LABL : { fprintf( yyout , "LABEL %s:" , argumento1 );
                                  break; }
        } /*switch*/
        fprintf( yyout , "\n" );
} /*gc*/


char *nuevaTemporal() {
        static int cont_tmp=1;
        char *t;

        t = (char *) malloc(10);
        sprintf( t , "temp%d" , cont_tmp );
        cont_tmp++;
        return t;
} /*nuevaTemporal*/


char *nuevaEtiqueta() {
        static int cont_etq=1;
        char *t;

        t = (char *) malloc(10);
        sprintf( t , "L%d" , cont_etq );
        cont_etq++;
        return t;
} /*nuevaEtiqueta*/
```

RESUMEN DE LA GRAMÁTICA

La gramática que se presenta a continuación reconoce un lenguaje de programación del estilo de C. Aunque se han añadido algunas instrucciones como por ejemplo la sentencia iterativa REPEAT y el bucle FOR clásico. Por ejemplo, debe reconocer código de este tipo:

```
repeat{
        ++i;
} until ( i>=j );

for i=0 to 100 step 5 do s += i;
```

Además cabe destacar que el bucle FOR es mucho más general que el existente en lenguajes como ***módula-2***.

La gramática queda de la siguiente manera:

```
sent            → '{' lista_sent '}'
                 | expresion ';'
                 | declaracion ';'
                 | cond ';'
                 | sent_switch
                 | sent_do ';'
                 | sent_while
                 | sent_repeat ';'
                 | sent_for
                 | sent_genuine_for
                 | sent_if

lista_sent      → lista_sent sent
                 | sent
```

Sentencias iterativas

```
sent_do         → DO sent WHILE '(' cond ')'

sent_while      → WHILE '(' cond ')' sent

sent_repeat     → REPEAT sent UNTIL '(' cond ')'

sent_for        → FOR '(' expresion ';' cond ';' expresion ')' sent

sent_genuine_for → FOR IDENT '=' expresion tos expresion estep DO sent

tos             → TO
                 | DOWNTO

estep           → STEP expresion
                 | ε
```

Declaración de variables

```
declaracion     → lista_decl IDENT
                 | lista_decl IDENT '[' ENTERO ']'

lista_decl      → lista_decl IDENT ','
                 | lista_decl IDENT '[' ENTERO ']' ','
                 | tipodato
```

```
tipodato        → CHAR
                | INT
                | REAL
```

Expresiones

```
expresion       → ENTERO
                | NREAL
                | IDENT
                | IDENT '[' expresion ']'
                | IDENT MASMAS
                | MASMAS IDENT
                | IDENT MENMEN
                | MENMEN IDENT
                | IDENT MASIG expresion
                | IDENT MENIG expresion
                | '(' expresion ')'
                | expresion '+' expresion
                | expresion '*' expresion
                | expresion '-' expresion
                | '-' expresion
                | expresion '/' expresion
                | expresion '%' expresion
                | IDENT '=' expresion
                | IDENT '[' expresion ']' '=' expresion
                | '(' cond ')' '?' expresion ':' expresion
```

Condiciones

```
cond            → expresion MENOR expresion
                | expresion MAYOR expresion
                | expresion IGIG expresion
                | expresion MAYORIG expresion
                | expresion MENORIG expresion
                | cond AND cond
                | cond OR cond
                | '(' cond ')'
                | NOT cond
                | TRUE
                | FALSE
```

Instrucciones selectivas

```
sent_switch     → cases DEFAULT ':' sent '}'
                | cases '}'

cases           → cases CASE expresion ':' sent break
                | SWITCH '(' expresion ')' '{'

break           → BREAK
                | ε

parte_if        → IF '(' cond ')' sent


elsifs          → parte_if
                | elsifs ELSIF '(' cond ')' sent

sent_if         → elsifs IF
                | elsifs ELSE sent
```

Esquema de traducción

El esquema de traducción queda de la siguiente forma:

gencod.y

```
%{

#include "tipos.h"
#include "gc.h"
#include "tablasim.h"

int RELATIVA=0;

%}


%union {
	char cadena[256];
	int existe;
	int direccion;
	struct{
		char cod[16];
		int tipo;
	} codigo;
	int tipo_decl;
	struct{
		char v[16];
		char f[16];
	} cond_estr;
	struct{
		char cod[16];
		char lbl[16];
		int tipo;
	} etq_interr;
	struct{
		char test[16];
		char actualizacion[16];
	} etq_for;
}

%right '='
%left '+'
%left '-'
%left '*' '/'
%left '%'

%token <cadena> REPEAT
%token <etq_for> FOR
%token SWITCH
%token DEFAULT
%token BREAK
%token UNTIL
%nonassoc <cadena> IF
%nonassoc ELSE
%nonassoc ELSIF
%token INT
%token REAL
%token CHAR
%token MASMAS
%token MENMEN
%token MASIG
%token MENIG
%token MAYOR
%token MENOR
%token MENORIG
%token MAYORIG
%token IGIG
%token <cadena> ENTERO
%token <cadena> NREAL
%token <cadena> IDENT
%token IGMAS
%token OR
%token AND
%token NOT
%token TRUE
%token FALSE
```

```
%left OR
%left AND

%nonassoc MASMAS
%nonassoc MENMEN
%nonassoc MASIG
%nonassoc MENIG
%nonassoc NOT
%nonassoc <etq_interr> '?'
%nonassoc ':'
%nonassoc '['
%nonassoc ']'
%nonassoc '{'
%nonassoc '}'
%nonassoc '('
%nonassoc ')'
%nonassoc <cadena> CASE

%type <codigo> expresion
%type <tipo_decl> lista_decl
%type <tipo_decl> tipodato
%type <cond_estr> cond
%type <etq_interr> cases
%type <existe> break
%type <etq_interr> elsifs
%type <etq_interr> parte_if
%token <cadena> DO
%token <cadena> WHILE
%token TO
%token STEP
%token DOWNTO
%type <codigo> estep
%type <direccion> tos

%%

/* --- Sentencias ---------------------------------------------------------- */

sent            : '{'                           {
                                                nuevobloque();
                                                }
                  lista_sent
                  '}'                           {
                                                finbloque();
                                                }
                | expresion ';'
                | declaracion ';'
                | cond ';'
                | sent_switch
                | sent_do ';'
                | sent_while
                | sent_repeat ';'
                | sent_for
                | sent_genuine_for
                | sent_if
                ;


/* --- Sentencias de iterativas -------------------------------------------- */

lista_sent      : lista_sent sent
                | sent
                ;


sent_do         : DO                            {
                                                strcpy( $1 , nuevaEtiqueta() );
                                                gc( C3D_LABL , $1 , "" , "" );
                                                }
                      sent WHILE '(' cond ')'   {
                                                gc( C3D_LABL , $6.v , "" , "" );
                                                gc( C3D_GOTO , "" , "" , $1 );
                                                gc( C3D_LABL , $6.f , "" , "" );
                                                }
                ;

sent_while      : WHILE                         {
                                                strcpy( $1 , nuevaEtiqueta() );
                                                gc( C3D_LABL , $1 , "" , "" );
                                                }
                      '(' cond ')'              {
                                                gc( C3D_LABL , $4.v , "" , "" );
```

```
                                              }
              sent                            {
                                              gc( C3D_GOTO , "" , "" , $1 );
                                              gc( C3D_LABL , $4.f , "" , "" );
                                              }
          ;


sent_repeat     : REPEAT                      {
                                              strcpy( $1 , nuevaEtiqueta() );
                                              gc( C3D_LABL , $1 , "" , "" );
                                              }
              sent UNTIL '(' cond ')'         {
                                              gc( C3D_LABL , $6.f , "" , "" );
                                              gc( C3D_GOTO , "" , "" , $1 );
                                              gc( C3D_LABL , $6.v , "" , "" );
                                              }
          ;

sent_for        : FOR '(' expresion ';'       {
                                              strcpy( $1.actualizacion , nuevaEtiqueta() );
                                              strcpy( $1.test , nuevaEtiqueta() );
                                              gc( C3D_LABL , $1.test , "" , "" );
                                              }
              cond ';'                        {
                                              gc( C3D_LABL , $1.actualizacion , "" , "" );
                                              }
              expresion ')'                   {
                                              gc( C3D_GOTO , "" , "" , $1.test );
                                              gc( C3D_LABL , $6.v , "" , "" );
                                              }
              sent                            {
                                              gc( C3D_GOTO , "" , "" , $1.actualizacion );
                                              gc( C3D_LABL , $6.f , "" , "" );
                                              }
          ;


sent_genuine_for: FOR IDENT '=' expresion     {
                                              NODO *extraido;
                                              // Primero vemos si IDENT esta en la tabla de
                                              // simbolos
                                              if ( !existe( $2 ) )
                                                     printf("ERROR: La variable '%s' no ha
                                                     sido definida previamente.\n",$2);
                                              else
                                                     extraido=buscar( $2 );
                                              // Comparamos el tipo de IDENT con expr
                                              if ( !comparartipos( (*extraido).tipo ,
                                                                   $4.tipo ) )
                                                     printf("ERROR: Asignacion de
                                                     '%s' erronea, comprueba los
                                                     tipos.\n",$2);
                                              // Ver si IDENT es un tipo simple
                                              if ( !tiposimple((*extraido).tipo) )
                                                     printf("ERROR: La variable '%s'
                                                     no es de tipo simple. Imposible
                                                     iterar sobre ella.\n",$2);
                                              gc( C3D_ASIG , $4.cod , "" , $2 );
                                              strcpy( $1.actualizacion ,
                                                     nuevaEtiqueta() );
                                              strcpy( $1.test , nuevaEtiqueta() );
                                              gc( C3D_LABL , $1.test , "" , "" );
                                              }
              tos expresion estep DO          {
                                              // Comparamos que los tipos coincidan
                                              if ( !comparartipos( $7.tipo , $4.tipo ) )
                                                     printf("ERROR: Asignacion de '%s'
                                                     erronea, comprueba los tipos.\n",$2);
                                              if ( !comparartipos( $8.tipo , $4.tipo ) )
                                                     printf("ERROR: Asignacion de '%s'
                                                     erronea, comprueba los tipos.\n",$2);

                                              if ( $6 == 1 )
                                                     gc(C3D_IFMA,$2,$7.cod,$1.actualizacion);
                                              else
                                                     gc(C3D_IFMA,$7.cod,$2,$1.actualizacion);
                                              }
              sent                            {
                                              if ( $6 == 1 )
                                                     gc( C3D_SUMA , $2 , $8.cod , $2 );
                                              else
```

```
                                                                gc( C3D_RSTA , $2 , $8.cod , $2 );
                                                                gc( C3D_GOTO , "" , "" , $1.test );
                                                                gc( C3D_LABL , $1.actualizacion ,"","");
                                                        }
                ;

tos             : TO                                    {$$=1;}
                | DOWNTO                                {$$=-1;}
                ;


estep           : STEP expresion                        {strcpy($$.cod,$2.cod);$$.tipo=$2.tipo;}
                |                                       {strcpy($$.cod,"1");$$.tipo=tipoint;}
                ;


/* --- Declaracion de variables ------------------------------------------- */

declaracion     : lista_decl IDENT              { //Recuerda: código casi idéntico a la regla de abajo
                                                NODO nuevo;

                                                if( existe( $2 ) ) // Miramos si ya está en la T.D.S
                                                        printf("ERROR, variable '%s' ya definida
                                                                previamente.\n",$2);
                                                else { // Entonces... lo metemos
                                                        strcpy( nuevo.nombre , $2 );
                                                        nuevo.direccion = RELATIVA;
                                                        nuevo.tam = tamtipo( $1 );
                                                        nuevo.tipo = $1;
                                                        insertar( nuevo );
                                                        RELATIVA = RELATIVA + tamtipo( $1 );
                                                } /*else*/
                                                }
                |  lista_decl IDENT '[' ENTERO ']'
                                                { //Recuerda: código casi idéntico a la regla de abajo
                                                NODO nuevo;

                                                if( existe( $2 ) ) // Miramos si ya está en la T.D.S.
                                                        printf("ERROR, variable '%s' ya definida
                                                                previamente.\n",$2);
                                                else { // Entonces... lo metemos
                                                        strcpy( nuevo.nombre , $2 );
                                                        nuevo.direccion = RELATIVA;
                                                        nuevo.tam = tamtipo( $1 )*atoi( $4 );
                                                        nuevo.tipo = componertipo( tipoarray , $1 );
                                                        insertar( nuevo );
                                                        RELATIVA = RELATIVA + tamtipo( $1 )*atoi($4);
                                                } /*else*/
                                                }
                ;

lista_decl      : lista_decl IDENT ','          {
                                                NODO nuevo;

                                                if( existe( $2 ) ) // Miramos si ya está en la T.D.S.
                                                        printf("ERROR, variable '%s' ya definida
                                                                previamente.\n",$2);
                                                else { // Entonces... lo metemos
                                                        strcpy( nuevo.nombre , $2 );
                                                        nuevo.direccion = RELATIVA;
                                                        nuevo.tam = tamtipo( $1 );
                                                        nuevo.tipo = $1;
                                                        insertar( nuevo );
                                                        RELATIVA = RELATIVA + tamtipo( $1 );
                                                } /*else*/
                                                $$ = $1;
                                                }
                | lista_decl IDENT '[' ENTERO ']' ','
                                                {
                                                NODO nuevo;

                                                if( existe( $2 ) ) // Miramos si ya está en la T.D.S.
                                                        printf("ERROR, variable '%s' ya definida
                                                                previamente.\n",$2);
                                                else { // Entonces... lo metemos
                                                        strcpy( nuevo.nombre , $2 );
                                                        nuevo.direccion = RELATIVA;
                                                        nuevo.tam = tamtipo($1)*atoi($4);
                                                        nuevo.tipo = componertipo( tipoarray , $1 );
                                                        insertar( nuevo );
                                                        RELATIVA = RELATIVA + tamtipo($1)*atoi($4);
                                                } /*else*/
```

```
                                $$ = $1;
                                }
                | tipodato      {$$ = $1;}
                ;

tipodato        : CHAR          {$$ = tipochar;}
                | INT           {$$ = tipoint;}
                | REAL          {$$ = tiporeal;}
                ;

/* --- Expresiones -------------------------------------------------------- */


expresion       : ENTERO        {
                                strcpy( $$.cod , $1 );
                                $$.tipo=tipoint;
                                }
                | NREAL         {
                                strcpy( $$.cod , $1 );
                                $$.tipo=tiporeal;
                                }
                | IDENT         {
                                NODO *extraido;
                                if ( !existe( $1 ) ) {
                                       printf("ERROR: La variable '%s' no ha sido
                                              definida previamente.\n",$1);
                                       $$.tipo=0;
                                } /*if*/
                                else{
                                       extraido=buscar( $1 );
                                      $$.tipo=(*extraido).tipo;
                                } /*else*/
                                strcpy( $$.cod , $1 );
                                }
                | IDENT '[' expresion ']'   {
                                NODO *extraido;
                                char *n;

                                n = nuevaTemporal();
                                if ( !existe( $1 ) ) {
                                printf("ERROR: La variable '%s' no ha sido definida
                                       previamente.\n",$1);
                                       $$.tipo=0;
                                } /*if*/
                                else{
                                       extraido=buscar( $1 );
                                       $$.tipo = tipobase( (*extraido).tipo );
                                } /*else*/
                                if ( $3.tipo != tipoint )
                                       printf("ERROR: El indice de un array debe ser un
                                       numero entero.\n");
                                gc( C3D_AARR , $1 , $3.cod , n );
                                strcpy( $$.cod , n );
                                }
                | IDENT MASMAS  {
                                char *n;
                                NODO *extraido;

                                if ( !existe( $1 ) )
                                       printf("ERROR: La variable '%s' no ha sido
                                       definida previamente.\n",$1);
                                else
                                       extraido=buscar( $1 );
                                n=nuevaTemporal();
                                gc( C3D_ASIG , $1 , "" , n );
                                gc( C3D_SUMA , $1 , "1" , $1 );
                                strcpy( $$.cod , n );
                                $$.tipo=(*extraido).tipo;
                                }
                | MASMAS IDENT  {
                                NODO *extraido;

                                if ( !existe( $2 ) )
                                       printf("ERROR: La variable '%s' no ha sido
                                              definida previamente.\n",$2);
                                else
                                       extraido=buscar( $2 );
                                gc( C3D_SUMA , $2 , "1" , $2 );
                                strcpy( $$.cod , $2 );
                                $$.tipo=(*extraido).tipo;
                                }
                | IDENT MENMEN  {
```

```
                            char *n;
                            NODO *extraido;

                            if ( !existe( $1 ) )
                                   printf("ERROR: La variable '%s' no ha sido
                                           definida previamente.\n",$1);
                            else
                                   extraido=buscar( $1 );
                            n=nuevaTemporal();
                            gc( C3D_ASIG , $1 , "" , n );
                            gc( C3D_RSTA , $1 , "1" , $1 );
                            strcpy( $$.cod , n );
                            $$.tipo=(*extraido).tipo;
                            }
| MENMEN IDENT              {
                            NODO *extraido;

                            if ( !existe( $2 ) )
                                   printf("ERROR: La variable '%s' no ha sido
                            definida previamente.\n",$2);
                            else
                                   extraido=buscar( $2 );
                            gc( C3D_RSTA , $2 , "1" , $2 );
                            strcpy( $$.cod , $2 );
                            $$.tipo=(*extraido).tipo;
                            }
| IDENT MASIG expresion     {
                            char *n;
                            NODO *extraido;

                            if ( !existe( $1 ) )
                                   printf("ERROR: La variable '%s' no ha sido
                                           definida previamente.\n",$1);
                            else
                                   extraido=buscar( $1 );
                            n=nuevaTemporal();
                            gc( C3D_SUMA , $1 , $3.cod , $1 );
                            strcpy( $$.cod , $1 );
                            if ( comparartipos( (*extraido).tipo , $3.tipo ) )
                                   $$.tipo = $3.tipo;
                            else {
                                   printf("ERROR: Las expresiones entran en
                                           conflicto de tipos.\n");
                                   $$.tipo = 0;
                            } /*else*/
                            }
| IDENT MENIG expresion     {
                            char *n;
                            NODO *extraido;

                            if ( !existe( $1 ) )
                                   printf("ERROR: La variable '%s' no ha sido
                                           definida previamente.\n",$1);
                            else
                                   extraido=buscar( $1 );
                            n=nuevaTemporal();
                            gc( C3D_RSTA , $1 , $3.cod , $1 );
                            strcpy( $$.cod , $1 );
                            if ( comparartipos( (*extraido).tipo , $3.tipo ) )
                                   $$.tipo = $3.tipo;
                            else {
                                   printf("ERROR: Las expresiones entran en
                                           conflicto de tipos.\n");
                                   $$.tipo = 0;
                            } /*else*/
                            }
| '(' expresion ')'         {
                                   strcpy( $$.cod , $2.cod );
                                   $$.tipo = $2.tipo;
                                   }
| expresion '+' expresion   {
                            if ( comparartipos( $1.tipo , $3.tipo ) )
                                   $$.tipo = $1.tipo;
                            else {
                                   printf("ERROR: Las expresiones entran en
                                           conflicto de tipos.\n");
                                   $$.tipo = 0;
                            } /*else*/
                            strcpy( $$.cod , nuevaTemporal() );
                            gc( C3D_SUMA , $1.cod , $3.cod , $$.cod );
                            }
```

```
| expresion '*' expresion   {
                            if ( comparartipos( $1.tipo , $3.tipo ) )
                                   $$.tipo = $1.tipo;
                            else {
                                   printf("ERROR: Las expresiones entran en
                                          conflicto de tipos.\n");
                                   $$.tipo = 0;
                            } /*else*/
                            strcpy( $$.cod , nuevaTemporal() );
                            gc( C3D_PROD , $1.cod , $3.cod , $$.cod );
                                   }
| expresion '-' expresion   {
                            if ( comparartipos( $1.tipo , $3.tipo ) )
                                   $$.tipo = $1.tipo;
                            else {
                                   printf("ERROR: Las expresiones entran en
                                          conflicto de tipos.\n");
                                   $$.tipo = 0;
                            } /*else*/
                            strcpy( $$.cod , nuevaTemporal() );
                            gc( C3D_RSTA , $1.cod , $3.cod , $$.cod );
                            }
| '-' expresion             {
                            char *t;

                            strcpy( $$.cod , nuevaTemporal() );
                            gc( C3D_RSTA , "0" , $2.cod , $$.cod );
                            $$.tipo=$2.tipo;
                            }
| expresion '/' expresion   {
                            if ( comparartipos( $1.tipo , $3.tipo ) )
                                   $$.tipo = $1.tipo;
                            else {
                                   printf("ERROR: Las expresiones entran en
                                          conflicto de tipos.\n");
                                   $$.tipo = 0;
                            } /*else*/
                            strcpy( $$.cod , nuevaTemporal() );
                            gc( C3D_DIVI , $1.cod , $3.cod , $$.cod );
                            }
| expresion '%' expresion {
                            char *n1,*n2;

                            if ( comparartipos( $1.tipo , $3.tipo ) )
                                   $$.tipo = $1.tipo;
                            else {
                                   printf("ERROR: Las expresiones entran en
                                          conflicto de tipos.\n");
                                   $$.tipo = 0;
                            } /*else*/
                                   n1=nuevaTemporal();
                            n2=nuevaTemporal();
                            strcpy( $$.cod , nuevaTemporal() );
                            gc( C3D_DIVI , $1.cod , $3.cod , n1 );
                            gc( C3D_PROD , n1 , $3.cod , n2 );
                            gc( C3D_RSTA , $1.cod , n2 , $$.cod );
                            }
|  IDENT '=' expresion      {
                            NODO *extraido;
                            // Primero vemos si IDENT esta en la tabla de simbolos
                            if ( !existe( $1 ) )
                                   printf("ERROR: La variable '%s' no ha sido
                                          definida previamente.\n",$1);
                            else
                                   extraido=buscar( $1 );
                            // Comparamos el tipo de IDENT con expresion
                            if ( !comparartipos( (*extraido).tipo , $3.tipo ) )
                                   printf("ERROR: Asignacion de '%s' erronea,
                                   comprueba los tipos.\n",$1);
                            gc( C3D_ASIG , $3.cod , "" , $1 );
                            $$.tipo=$3.tipo;
                            strcpy( $$.cod , $3.cod );
                            }
|  IDENT '[' expresion ']' '=' expresion
                            {
                            NODO *extraido;
                            char *n;

                            n = nuevaTemporal();
                            // Primero vemos si IDENT esta en la tabla de simbolos
                            if ( !existe( $1 ) ) {
```

```
                                        printf("ERROR: La variable '%s' no ha sido
                                                definida previamente.\n",$1);
                                        $$.tipo=0;
                                } /*if*/
                                else{
                                        extraido=buscar( $1 );
                                        $$.tipo = tipobase( (*extraido).tipo );
                                } /*else*/
                                if ( $3.tipo != tipoint )
                                        printf("ERROR: El indice de un array debe ser un
                                                numero entero.\n");
                                // Comparamos el tipo de IDENT con expresion
                                if ( !comparartipos(tipobase((*extraido).tipo),$6.tipo))
                                        printf("ERROR: Asignacion de '%s' erronea,
                                                        comprueba los tipos.\n",$1);
                                gc( C3D_ARRA , $3.cod , $6.cod , $1 );
                                $$.tipo=$6.tipo;
                                strcpy( $$.cod , $6.cod );
                                }
                | '(' cond ')' '?'      {
                                strcpy( $4.cod , nuevaTemporal() );
                                strcpy( $4.lbl , nuevaEtiqueta() );
                                gc( C3D_LABL , $2.v , "" , "" );
                                }
                        expresion       {
                                gc( C3D_ASIG , $6.cod , "" , $4.cod );
                                gc( C3D_GOTO , "" , "" , $4.lbl );
                                gc( C3D_LABL , $2.f , "" , "" );
                                }
                        ':' expresion   {
                                gc( C3D_ASIG , $9.cod , "" , $4.cod );
                                gc( C3D_LABL , $4.lbl , "" , "" );
                                if ( comparartipos( $6.tipo , $9.tipo ) )
                                        $$.tipo = $6.tipo;
                                else {
                                        printf("ERROR: Las expresiones entran en
                                                conflicto de tipos.\n");
                                        $$.tipo = 0;
                                } /*else*/
                                strcpy( $$.cod , $4.cod );
                                }
                ;

/* --- Condiciones -------------------------------------------------------- */

cond            : expresion MENOR expresion     {
                                        char *l1,*l2;

                                        if ( !comparartipos( $1.tipo , $3.tipo ) )
                                                printf("ERROR: Las expresiones entran en
                                                        conflicto de tipos y no se
                                                        pueden comparar.\n");
                                        l1=nuevaEtiqueta();
                                        l2=nuevaEtiqueta();
                                        gc( C3D_IFMA , $3.cod , $1.cod , l1 );
                                        gc( C3D_GOTO , "" , "" , l2 );
                                        strcpy( $$.v , l1 );
                                        strcpy( $$.f , l2 );
                                        }
                | expresion MAYOR expresion     {
                                        char *l1,*l2;

                                        if ( !comparartipos( $1.tipo , $3.tipo ) )
                                                printf("ERROR: Las expresiones entran en
                                                        conflicto de tipos y no se
                                                        pueden comparar.\n");
                                        l1=nuevaEtiqueta();
                                        l2=nuevaEtiqueta();
                                        gc( C3D_IFMA , $1.cod , $3.cod , l1 );
                                        gc( C3D_GOTO , "" , "" , l2 );
                                        strcpy( $$.v , l1 );
                                        strcpy( $$.f , l2 );
                                        }
                | expresion IGIG expresion      {
                                        char *l1,*l2;

                                        if ( !comparartipos( $1.tipo , $3.tipo ) )
                                                printf("ERROR: Las expresiones entran en
                                                        conflicto de tipos y no se
                                                        pueden comparar.\n");
                                        l1=nuevaEtiqueta();
                                        l2=nuevaEtiqueta();
```

```
                                        gc( C3D_IFIG , $1.cod , $3.cod , l1 );
                                        gc( C3D_GOTO , "" , "" , l2 );
                                        strcpy( $$.v , l1 );
                                        strcpy( $$.f , l2 );
                                        }
          | expresion MAYORIG expresion {
                                                char *l1,*l2;

                                        if ( !comparartipos( $1.tipo , $3.tipo ) )
                                                printf("ERROR: Las expresiones
                                                        entran en conflicto de tipos y no
                                                                se pueden comparar.\n");
                                        l1=nuevaEtiqueta();
                                        l2=nuevaEtiqueta();
                                        gc( C3D_IFMA , $1.cod , $3.cod , l1 );
                                        gc( C3D_IFIG , $1.cod , $3.cod , l1 );
                                        gc( C3D_GOTO , "" , "" , l2 );
                                        strcpy( $$.v , l1 );
                                        strcpy( $$.f , l2 );
                                        }
          | expresion MENORIG expresion {
                                        char *l1,*l2;

                                        if ( !comparartipos( $1.tipo , $3.tipo ) )
                                                printf("ERROR: Las expresiones entran en
                                                        conflicto de tipos y no se
                                                        pueden comparar.\n");
                                        l1=nuevaEtiqueta();
                                        l2=nuevaEtiqueta();
                                        gc( C3D_IFMA , $1.cod , $3.cod , l1 );
                                        gc( C3D_GOTO , "" , "" , l2 );
                                        strcpy( $$.v , l2 );
                                        strcpy( $$.f , l1 );
                                        }
          | cond AND                    {
                                        gc( C3D_LABL , $1.v , "" ,"" );
                                        }
                cond                    {
                                        gc( C3D_LABL , $1.f , "" , "" );
                                        gc( C3D_GOTO , "" , "" , $4.f );
                                        strcpy( $$.v , $4.v );
                                        strcpy( $$.f , $4.f );
                                        }
          | cond OR                     {
                                        gc( C3D_LABL , $1.f , "" ,"" );
                                        }
                cond                            {
                                        gc( C3D_LABL , $1.v , "" , "" );
                                        gc( C3D_GOTO , "" , "" , $4.v );
                                        strcpy( $$.v , $4.v );
                                        strcpy( $$.f , $4.f );
                                        }
          | '(' cond ')'                {
                                        strcpy( $$.v , $2.v );
                                        strcpy( $$.f , $2.f );
                                        }
          | NOT cond                    {
                                        strcpy( $$.v , $2.f );
                                        strcpy( $$.f , $2.v );
                                        }
          | TRUE                        {
                                        strcpy( $$.v , nuevaEtiqueta() );
                                        strcpy( $$.f , nuevaEtiqueta() );
                                        gc( C3D_GOTO , "" , "" , $$.v );
                                        }
          | FALSE                       {
                                        strcpy( $$.v , nuevaEtiqueta() );
                                        strcpy( $$.f , nuevaEtiqueta() );
                                        gc( C3D_GOTO , "" , "" , $$.f );
                                        }
          ;

/* --- Selectivas ---------------------------------------------------------- */

sent_switch : cases DEFAULT ':' sent '}'        {
                                        gc( C3D_LABL , $1.lbl , "" , "");
                                        }
          | cases '}'                   {
                                        gc( C3D_LABL , $1.lbl , "" , "");
                                        }
          ;
```

```
cases           : cases CASE expresion ':'         {
                                                   char *L1,*L2;

                                                   if ( !comparartipos( $1.tipo , $3.tipo ) )
                                                           printf("ERROR: Las expresiones entran en
                                                                   conflicto de tipos y no se
                                                                   pueden comparar.\n");
                                                   L1 = nuevaEtiqueta();
                                                   L2 = nuevaEtiqueta();
                                                   gc( C3D_IFIG , $1.cod , $3.cod , L1 );
                                                   gc( C3D_GOTO , "" , "" , L2 );
                                                   gc( C3D_LABL , L1 , "" , "" );
                                                   strcpy( $2 , L2 );
                                                   }
                      sent   break                 {
                                                   if ($7)
                                                           gc( C3D_GOTO , "" , "" , $1.lbl );
                                                   gc( C3D_LABL , $2 , "" , "" );
                                                   strcpy( $$.lbl , $1.lbl );
                                                   strcpy( $$.cod , $1.cod );
                                                   $$.tipo = $1.tipo;
                                                   }
                | SWITCH '(' expresion ')' '{'     {
                                                   char *L;

                                                   L = nuevaEtiqueta();
                                                   strcpy( $$.cod , $3.cod );
                                                   strcpy( $$.lbl , L );
                                                   $$.tipo = $3.tipo;
                                                   }
                ;

break           : BREAK                            {$$ = 1;}
                |                                  {$$ = 0;}
                ;

parte_if        : IF '(' cond ')'                  {
                                                   gc( C3D_LABL , $3.v , "" , "" );
                                                   }
                      sent                         {
                                                   char *L;
                                                   L = nuevaEtiqueta();
                                                   gc( C3D_GOTO , "" , "" , L );
                                                   gc( C3D_LABL , $3.f , "" , "" );
                                                   strcpy( $$.lbl , L );
                                                   }
                ;


elsifs          : parte_if                         {
                                                   strcpy( $$.lbl , $1.lbl );
                                                   }
                | elsifs ELSIF '(' cond ')'        {
                                                   gc( C3D_LABL , $4.v , "" , "" );
                                                   }
                      sent                         {
                                                   gc( C3D_GOTO , "" , "" , $1.lbl );
                                                   gc( C3D_LABL , $4.f , "" , "" );
                                                   }
                ;


sent_if         : elsifs %prec IF                  {
                                                   gc( C3D_LABL , $1.lbl , "" , "" );
                                                   }
                | elsifs ELSE sent                 {
                                                   gc( C3D_LABL , $1.lbl , "" , "" );
                                                   }
                ;

%%
```

La función principal

La función siguiente se encarga de llamar al **parser** de *yacc*. En el fichero se ha incluido 2 funciones necesarias para decir qué hacer cuando se llegue a fín de fichero y otra función para decir qué hacer si *yacc* no puede reducir por ninguna regla.

main.c

```
#include <stdio.h>
#include "tablasim.h"

extern FILE *yyout;
extern FILE *yyin;


int main( int argc , char **argv ){

        if ( (argc==2) || (argc==3) ) {
                yyin = fopen( argv[1], "r" );
                if ( yyin == NULL ) {
                        printf("\nERROR: El fichero '%s' no se puede abrir en lectura.\n",argv[1]);
                        return(-1);
                } /*if*/
                if ( argc==3 ) {
                        yyout = fopen( argv[2], "w" );
                        if ( yyout == NULL ) {
                                printf("\nERROR: El fichero '%s' no se puede crear.\n",argv[1]);
                                return(-1);
                        } /*if*/
                } /*if*/
        } /*if*/
        else {
                printf("\nAtencion: Se utilizara la E/S estandar.\n");
        } /*else*/

        inicializar_tabla();  // Iniciamos la tabla de símbolos
        yyparse();
        // Deberiamos destruir la tabla... es algo pendiente
        return(0);
} /* main */


int yywrap(){

        return(1);
} /* yywrap */


void yyerror(char* st) {

        printf("Error detectado durante el analisis:\n\t\"%s\"\n",st);
} /*yyerror*/
```

El Makefile y características del proyecto

Para compilar todo el proyecto es conveniente el uso de un fichero *makefile*. Para el desarrollo se ha empleado el entorno DJGPP (Es un entorno para Windows que contiene todos los compiladores de GNU, como el gcc, así como lex y yacc). La portabilidad entre linux y windows es inmediata, en parte debido a que cumple con el estándar ANSI-C. Por tanto, con sólo modificar el *makefile* tenemos el proyecto funcionando en ambas plataformas. Como complemento, aunque es indispensable, se ha usado el cygwin, para tener un shell indéntico que el existente en linux (y por extensión a UNIX) con el mismo repertorio de instrucciones.

Makefile

```
# Reglas de compilación

gencod.exe:    tablasim.o tipos.o gc.o y.tab.o lexyy.o main.o
               gcc tablasim.o tipos.o gc.o main.o lexyy.o y.tab.o -o gencod.exe

tablasim.o:    tablasim.c tablasim.h
               gcc -c tablasim.c

tipos.o:       tipos.c tipos.h
               gcc -c tipos.c

gc.o:          gc.c gc.h
               gcc -c gc.c

y.tab.o:       y.tab.c
               gcc -c y.tab.c

y.tab.c:       gencod.y
               yacc -d gencod.y

lexyy.o:       lexyy.c
               gcc -c lexyy.c

lexyy.c:       gencod.l
               lex gencod.l

main.o: main.c
               gcc -c main.c

clean:
               rm -r gencod.exe lexyy.c lexyy.o main.o y.tab.c y.tab.h y.tab.o gc.o tablasim.o
tipos.o

run:           gencod.exe
               gencod.exe test.txt | less
```