```
/*Lucas Gallego Bravo Grupo: 93 */
/*100429005@alumnos.uc3m.es*/


%{                      // SECCION 1 Declaraciones de C-Yacc

#include <stdio.h>
#include <ctype.h>           // declaraciones para tolower
#include <string.h>          // declaraciones para cadenas
#include <stdlib.h>          // declaraciones para exit ()

#define FF fflush(stdout);    // para forzar la impresion inmediata

int yylex () ;
int yyerror () ;
char *mi_malloc (int) ;
char *gen_code (char *) ;
char *int_to_string (int) ;
char *char_to_string (char) ;

/*Funciones creadas para el uso de Array_Local*/
void cleanArray(char*[]);
int searchArray(char*[],char*);

char temp [2048] ;
char name_func [64];  /*En este char se guardara el nombre de las funciones*/
char *array_local[50];  /*Array para las variables locales*/
int index_local = 0;  /*Index para el array*/

// Definitions for explicit attributes
```

```
typedef struct s_attr {
        int value ;
        char *code ;
} t_attr ;

#define YYSTYPE t_attr

%}

// Definitions for explicit attributes

%token NUMBER
%token IDENTIF       // Identificador=variable
%token INTEGER       // identifica el tipo entero
%token STRING
%token MAIN          // identifica el comienzo del proc. main
%token WHILE         // identifica el bucle main
%token PUTS          // identifica el comando puts
%token PRINTF        // identifica el comando printf
%token OR
%token AND
%token EQUAL
%token DIFF
%token SMALLER
%token BIGGER
%token FOR
%token RETURN
%token IF
%token ELSE

// Definicion de prioridades de los operadores
```

```
%right '='
%left OR
%left AND
%left DIFF EQUAL
%left '<' SMALLER '>' BIGGER
%left '+' '-'
%left '*' '/' '%'
%left UNARY_SIGN '!'

%%                          // Seccion 3 Gramatica - Semantico

axioma:       decl_var decl_fun      { strcpy(temp,"");
                                       strcat(temp, $1.code) ;
                                       strcat(temp, $2.code) ;
                                       printf("%s\n",temp);}
          ;


decl_var:   /*lambda*/                 { strcpy(temp,"");
                                     $$.code = gen_code (temp) ; }

          |   global ';' decl_var  {   sprintf (temp, "%s%s", $1.code, $3.code) ;
                                     $$.code = gen_code (temp) ;}
          ;


global:      INTEGER setq_glob              {sprintf (temp, "%s", $2.code);
                                        strcat(temp,"\n");
                                        $$.code = gen_code (temp) ; }
          ;
```

```
setq_glob:  IDENTIF          {  strcpy(temp,"");
                              sprintf (temp, "(setq %s 0)", $1.code) ;
                              $$.code = gen_code (temp) ;}

          |  IDENTIF '=' termino     {sprintf (temp, "(setq %s %s)", $1.code,  $3.code) ;
                                    $$.code = gen_code (temp) ;}

          |  IDENTIF '[' operando ']'  {   strcpy(temp,"");
                                       sprintf(temp, "(setq %s (make-array %s))",$1.code,$3.code);
                                       $$.code = gen_code (temp) ;}

          |   IDENTIF ',' setq_glob          {sprintf (temp, "(setq %s 0)", $1.code) ;
                                         strcat (temp,$3.code);
                                         $$.code = gen_code (temp) ;}

          |  IDENTIF '=' termino ',' setq_glob   { sprintf (temp, "(setq %s %s)", $1.code,  $3.code) ;
                                            strcat (temp,$5.code);
                                                 $$.code = gen_code (temp) ;}

          |  IDENTIF '[' operando ']' ',' setq_glob    {    strcpy(temp,"");
                                                  sprintf(temp, "(setq %s (make-array %s))",$1.code,$3.code);
                                                  strcat (temp,$5.code);
                                                  $$.code = gen_code (temp) ;}
          ;


decl_fun:   funciones   main              {   strcpy(temp,"");
                                        strcat(temp, $1.code);
                                        cleanArray(array_local);
                                        strcat(temp,$2.code);
```

```
                                $$.code = gen_code (temp) ;}
        ;


funciones:                      {  strcpy(temp,"");
                                $$.code = gen_code (temp) ;}

        | IDENTIF                { strcpy(name_func,$1.code);}

           '(' argumentos_func')' '{' cuerpo '}' funciones      {    strcpy(temp,"");
                                                        sprintf(temp,"(defun %s (%s)\n",$1.code,$4.code);
                                                        strcat(temp, $7.code);
                                                        strcat(temp,"\n");
                                                        strcat(temp,")\n");
                                                        cleanArray(array_local);
                                                        strcpy(name_func,"");
                                                        strcat(temp,$9.code);
                                                        $$.code = gen_code (temp) ;}
        ;


main:       MAIN                { sprintf(name_func,"%s",$1.code); }

        '(' argumentos_func')' '{' cuerpo '}'     {    strcpy(temp,"");
                                            sprintf(temp,"(defun main (%s)\n",$4.code);
                                            strcat(temp, $7.code);
                                            strcat(temp,"\n");
                                            strcat(temp,")");
                                            $$.code = gen_code (temp) ;}
        ;
```

```
cuerpo:     PRINTF '(' imprimir  ')'  ';'      {   strcpy(temp,"");
                                        strcat(temp,$3.code);
                                        $$.code = gen_code (temp) ; }

          | PUTS '(' STRING ')' ';'          { strcpy(temp,"");
                                        sprintf(temp, "(print \"%s\")",$3.code);
                                        $$.code = gen_code (temp) ;}

          | IDENTIF '=' expresion  ';'    { strcpy(temp,"");
                                      if (searchArray(array_local, $1.code) == 0){
                                        sprintf (temp, "(setf %s_%s %s)", name_func,$1.code, $3.code) ;
                                      }
                                      else{
                                        sprintf (temp, "(setf %s %s)", $1.code, $3.code) ;
                                      }
                                      $$.code = gen_code (temp) ; }

          | INTEGER setq ';'              { strcpy(temp,"");
                                      strcat(temp,$2.code);
                                      $$.code = gen_code (temp) ;}

          | WHILE '(' expresion ')' '{' cuerpo '}' {  strcpy(temp,"");
                                          sprintf (temp, "(loop while %s do %s)", $3.code, $6.code) ;
                                          $$.code = gen_code (temp) ;}

          | FOR '(' init ';' expresion ';' inc_dec ')' '{' cuerpo '}'  {  strcpy(temp,"");
                                                    sprintf (temp, "%s\n(loop while %s do %s \n%s)", $3.code,$5.code, $10.code, $7.code);
                                                    $$.code = gen_code (temp) ;}

          | IF '(' expresion ')' '{' cuerpo '}'      { strcpy(temp,"");
```

```
                                    sprintf(temp,"(if %s\n(progn %s))",$3.code,$6.code);
                                    $$.code = gen_code (temp) ;}

| IF '(' expresion ')' '{' cuerpo '}' ELSE '{' cuerpo '}'    { strcpy(temp,"");
                                                  sprintf(temp,"(if %s\n(progn %s)\n(progn %s))",$3.code,$6.code, $10.code);
                                                  $$.code = gen_code (temp) ;}

| IDENTIF '('parametros')' ';'          { strcpy(temp,"");
                                  sprintf(temp,"(%s %s)",$1.code,$3.code);
                                  $$.code = gen_code (temp) ;}

| RETURN expresion ';'               {strcpy(temp,"");
                                sprintf(temp,"(return-from %s %s)",name_func,$2.code);
                                $$.code = gen_code (temp) ;}

| IDENTIF '['expresion']' '=' expresion ';'  { strcpy(temp,"");
                                     if (searchArray(array_local, $1.code) == 0){
                                      sprintf(temp,"(setf (aref %s_%s %s) %s)",name_func,$1.code,$3.code,$6.code);
                                     }
                                     else{
                                      sprintf(temp,"(setf (aref %s %s) %s)",$1.code,$3.code,$6.code);
                                     }
                                     $$.code = gen_code (temp) ;}

| PRINTF '(' imprimir  ')'  ';'  cuerpo    {   strcpy(temp,"");
                                      strcat(temp, $3.code);
                                      strcat(temp,"\n");
                                      strcat(temp, $6.code);
                                      $$.code = gen_code (temp) ; }

| PUTS '(' STRING ')' ';'  cuerpo        {  strcpy(temp,"");
```

```
                                    sprintf(temp, "(print \"%s\")",$3.code);
                                    strcat(temp,"\n");
                                    strcat(temp, $6.code);
                                    $$.code = gen_code (temp) ;}

| IDENTIF '=' expresion  ';'  cuerpo  { strcpy(temp,"");
                                    if (searchArray(array_local, $1.code) == 0){
                                    sprintf (temp, "(setf %s_%s %s)", name_func,$1.code, $3.code) ;
                                    }
                                    else{
                                    sprintf (temp, "(setf %s %s)", $1.code, $3.code) ;
                                    }
                                    strcat(temp,"\n");
                                    strcat(temp, $5.code);
                                    $$.code = gen_code (temp) ; }

| INTEGER setq ';'  cuerpo       {  strcpy(temp,"");
                                strcat(temp, $2.code);
                                strcat(temp,"\n");
                                strcat(temp, $4.code);
                                $$.code = gen_code (temp) ;}

| WHILE '(' expresion ')' '{' cuerpo '}' cuerpo   { strcpy(temp,"");
                                        sprintf (temp, "(loop while %s do %s)\n", $3.code, $6.code) ;
                                        strcat(temp,$8.code);
                                        $$.code = gen_code (temp) ;}

| FOR '(' init ';' expresion ';' inc_dec ')' '{' cuerpo '}' cuerpo  { strcpy(temp,"");
                                                sprintf (temp, "%s\n(loop while %s do %s \n%s)", $3.code,$5.code, $10.code, $7.code) ;
                                                strcat(temp,$12.code);
                                                $$.code = gen_code (temp) ;}
```

```
        | IF '(' expresion ')' '{' cuerpo '}'  cuerpo    { strcpy(temp,"");
                                                sprintf(temp,"(if %s\n(progn %s))",$3.code,$6.code);
                                                strcat(temp,$8.code);
                                                $$.code = gen_code (temp) ;}

        | IF '(' expresion ')' '{' cuerpo '}' ELSE '{' cuerpo '}' cuerpo  { strcpy(temp,"");
                                                        sprintf(temp,"(if %s\n(progn %s)\n(progn %s))",$3.code,$6.code, $10.code);
                                                        strcat(temp,$12.code);
                                                        $$.code = gen_code (temp) ;}

        | IDENTIF '('parametros')' ';' cuerpo            { strcpy(temp,"");
                                                sprintf(temp,"(%s %s)\n%s",$1.code,$3.code,$6.code);
                                                $$.code = gen_code (temp) ;}

        | IDENTIF '['expresion']' '=' expresion ';' cuerpo { strcpy(temp,"");
                                                if (searchArray(array_local, $1.code) == 0){
                                                   sprintf(temp,"(setf (aref %s_%s %s) %s)",name_func,$1.code,$3.code,$6.code);
                                                }
                                                else{
                                                   sprintf(temp,"(setf (aref %s %s) %s)",$1.code,$3.code,$6.code);
                                                }
                                                strcat(temp,$8.code);
                                                $$.code = gen_code (temp) ;}
        ;


init:     IDENTIF '=' expresion              {  strcpy(temp,"");
                                    sprintf (temp, "(setf %s_%s %s)", name_func,$1.code,$3.code) ;
                                    $$.code = gen_code (temp) ;}
```

```
        | INTEGER IDENTIF '=' expresion     {  strcpy(temp,"");
                                              sprintf (temp, "(setq %s_%s %s)", name_func,$1.code,  $3.code) ;
                                              $$.code = gen_code (temp) ;}
        ;


inc_dec:    IDENTIF '=' expresion          { strcpy(temp,"");
                                             sprintf (temp, "(setf %s_%s %s)", name_func,$1.code, $3.code) ;
                                             $$.code = gen_code (temp) ; }
        ;


imprimir:   STRING ',' imprimir2      {   sprintf (temp, "%s", $3.code);
                                          $$.code = gen_code (temp) ; }
        ;


imprimir2:    expresion             { sprintf (temp, "(prin1 %s)", $1.code);
                                      $$.code = gen_code (temp) ;      }

        | STRING                    { sprintf (temp, "(prin1 \"%s\") ", $1.code);
                                    $$.code = gen_code (temp) ; }

        | expresion ',' imprimir2 {  sprintf (temp, "(prin1 %s) ", $1.code);
                                     strcat (temp,$3.code);
                                     $$.code = gen_code (temp) ; }

        | STRING ',' imprimir2      {   sprintf (temp, "(prin1 \"%s\") ", $1.code);
                                        strcat (temp,$3.code);
                                        $$.code = gen_code (temp) ; }
        ;
```

```
argumentos_func:                                {strcpy(temp,"");
                                              $$.code = gen_code (temp) ;}

              | INTEGER IDENTIF                    {  strcpy(temp,"");
                                                  sprintf (temp, "%s ",$2.code) ;
                                                  $$.code = gen_code (temp) ;}

              | INTEGER IDENTIF ',' argumentos_func2   {  strcpy(temp,"");
                                                  sprintf (temp, "%s ",$2.code) ;
                                                  strcat(temp,$4.code);
                                                  $$.code = gen_code (temp) ;}
              ;


argumentos_func2:     INTEGER IDENTIF                   {  strcpy(temp,"");
                                                  sprintf (temp, "%s ",$2.code) ;
                                                  $$.code = gen_code (temp) ;}

              | INTEGER IDENTIF ',' argumentos_func2   {  strcpy(temp,"");
                                                  sprintf (temp, "%s ",$2.code) ;
                                                  strcat(temp,$4.code);
                                                  $$.code = gen_code (temp) ;}
              ;


parametros:                 {  strcpy(temp,"");
                          $$.code = gen_code (temp) ;}

            | expresion      {  strcpy(temp,"");
```

```
                        sprintf(temp,"%s",$1.code);
                        $$.code = gen_code (temp) ;}

        | expresion ',' parametros2      {  strcpy(temp,"");
                                           sprintf(temp,"%s ",$1.code);
                                           strcat(temp,$3.code);
                                           $$.code = gen_code (temp) ;}
        ;


parametros2:    expresion       {  strcpy(temp,"");
                                   sprintf(temp,"%s",$1.code);
                                   $$.code = gen_code (temp) ;}

        | expresion ',' parametros2      {  strcpy(temp,"");
                                           sprintf(temp,"%s ",$1.code);
                                           strcat(temp,$3.code);
                                           $$.code = gen_code (temp) ;}
        ;


setq:       IDENTIF          {  strcpy(temp,"");
                                array_local[index_local] =  $1.code;
                                index_local += 1;
                                sprintf (temp, "(setq %s_%s 0)", name_func,$1.code) ;
                                $$.code = gen_code (temp) ;}

      |  IDENTIF '=' termino     {sprintf (temp, "(setq %s_%s %s)", name_func,$1.code,  $3.code) ;
                                   array_local[index_local] =  $1.code;
                                   index_local += 1;
                                    $$.code = gen_code (temp) ;}
```

```
        |  IDENTIF '[' expresion ']'  {  strcpy(temp,"");
                                sprintf(temp, "(setq %s (make-array %s))",$1.code,$3.code);
                                array_local[index_local] =  $1.code;
                                index_local += 1;
                                $$.code = gen_code (temp) ;}

        |   IDENTIF ',' setq          {sprintf (temp, "(setq %s_%s 0)", name_func,$1.code) ;
                                array_local[index_local] =  $1.code;
                                index_local += 1;
                                strcat (temp,$3.code);
                                $$.code = gen_code (temp) ;}

        |  IDENTIF '=' termino ',' setq   {sprintf (temp, "(setq %s_%s %s)", name_func,$1.code,  $3.code) ;
                                    array_local[index_local] =  $1.code;
                                    index_local += 1;
                                    strcat (temp,$5.code);
                                    $$.code = gen_code (temp) ;}

        |  IDENTIF '[' expresion ']' ',' setq    {   strcpy(temp,"");
                                       sprintf(temp, "(setq %s (make-array %s))",$1.code,$3.code);
                                       array_local[index_local] =  $1.code;
                                       index_local += 1;
                                       strcat (temp,$5.code);
                                       $$.code = gen_code (temp) ;}
        ;


expresion:      termino                 { $$ = $1 ; }

        |   expresion '+' expresion  { sprintf (temp, "(+ %s %s)", $1.code, $3.code) ;
```

```
                        $$.code = gen_code (temp) ; }

|   expresion '-' expresion  { sprintf (temp, "(- %s %s)", $1.code, $3.code) ;
                        $$.code = gen_code (temp) ; }

|   expresion '*' expresion  { sprintf (temp, "(* %s %s)", $1.code, $3.code) ;
                        $$.code = gen_code (temp) ; }

|   expresion '/' expresion  { sprintf (temp, "(/ %s %s)", $1.code, $3.code) ;
                        $$.code = gen_code (temp) ; }

|   expresion OR expresion   { sprintf (temp, "(or %s %s)", $1.code, $3.code) ;
                        $$.code = gen_code (temp) ; }

|   expresion AND expresion   { sprintf (temp, "(and %s %s)", $1.code, $3.code) ;
                        $$.code = gen_code (temp) ; }

|   expresion DIFF expresion   { sprintf (temp, "(/= %s %s)", $1.code, $3.code) ;
                         $$.code = gen_code (temp) ; }

|   expresion EQUAL expresion   { sprintf (temp, "(= %s %s)", $1.code, $3.code) ;
                         $$.code = gen_code (temp) ; }

|   expresion '<' expresion   { sprintf (temp, "(< %s %s)", $1.code, $3.code) ;
                         $$.code = gen_code (temp) ; }

|   expresion '>' expresion   { sprintf (temp, "(> %s %s)", $1.code, $3.code) ;
                         $$.code = gen_code (temp) ; }

|   expresion SMALLER expresion   { sprintf (temp, "(<= %s %s)", $1.code, $3.code) ;
                         $$.code = gen_code (temp) ; }
```

```
        |   expresion BIGGER expresion   { sprintf (temp, "(>= %s %s)", $1.code, $3.code) ;
                                           $$.code = gen_code (temp) ; }

        |   expresion '!' expresion   { sprintf (temp, "(not %s %s)", $1.code, $3.code) ;
                                           $$.code = gen_code (temp) ; }

        |   expresion '%' expresion   { sprintf (temp, "(mod %s %s)", $1.code, $3.code) ;
                                           $$.code = gen_code (temp) ; }
        ;


termino:       operando                          { $$ = $1 ; }
        |   '+' operando %prec UNARY_SIGN      { sprintf (temp, "(+ %s)", $2.code) ;
                                           $$.code = gen_code (temp) ; }
        |   '-' operando %prec UNARY_SIGN      { sprintf (temp, "(- %s)", $2.code) ;
                                           $$.code = gen_code (temp) ; }
        ;


operando:      IDENTIF                  {  if (searchArray(array_local, $1.code) == 0){
                                          sprintf (temp, "%s_%s",name_func,$1.code) ;
                                       }
                                       else{
                                          sprintf (temp, "%s", $1.code) ;
                                       }
                                       $$.code = gen_code (temp) ; }

        |   NUMBER                    { sprintf (temp, "%d", $1.value) ;
                                     $$.code = gen_code (temp) ; }
        |   '(' expresion ')'       { $$ = $2 ; }
```

```
        | IDENTIF '('parametros')'   { strcpy(temp,"");
                                     sprintf(temp,"(%s %s)",$1.code,$3.code);
                                     $$.code = gen_code (temp) ;}

        | IDENTIF '['expresion']'       {  if (searchArray(array_local, $1.code) == 0){
                                            sprintf (temp, "(aref %s_%s %s)",name_func,$1.code,$3.code) ;
                                           }
                                           else{
                                            sprintf (temp, "(aref %s %s)",$1.code,$3.code);
                                           }
                                           $$.code = gen_code (temp) ; }
        ;


%%                          // SECCION 4    Codigo en C

int n_line = 1 ;

int yyerror (mensaje)
char *mensaje ;
{
   fprintf (stderr, "%s en la linea %d\n", mensaje, n_line) ;
   printf ( "\n") ;           // bye
}

char *int_to_string (int n)
{
   sprintf (temp, "%d", n) ;
   return gen_code (temp) ;
}
```

```
char *char_to_string (char c)
{
   sprintf (temp, "%c", c) ;
   return gen_code (temp) ;
}

char *my_malloc (int nbytes)       // reserva n bytes de memoria dinamica
{
   char *p ;
   static long int nb = 0;        // sirven para contabilizar la memoria
   static int nv = 0 ;            // solicitada en total

   p = malloc (nbytes) ;
   if (p == NULL) {
      fprintf (stderr, "No queda memoria para %d bytes mas\n", nbytes) ;
      fprintf (stderr, "Reservados %ld bytes en %d llamadas\n", nb, nv) ;
      exit (0) ;
   }
   nb += (long) nbytes ;
   nv++ ;

   return p ;
}
/*****************************************************************************/
/**************** Seccion de Funciones creadas Array_Local *****************/
/*****************************************************************************/

void cleanArray(char *array[]) {
   for (int i = 0; i < 50; i++) {
      array[i] = NULL;
```

```
    }
}

int searchArray(char *array[], char *target) {
    for (int i = 0; i < 50; i++) {
        if (array[i] != NULL && strcmp(array[i], target) == 0) {
            return 0;
        }
    }
    return 1; // Return 1 if the target is not found
}

/*****************************************************************************/
/********************** Seccion de Palabras Reservadas **********************/
/*****************************************************************************/

typedef struct s_keyword { // para las palabras reservadas de C
    char *name ;
    int token ;
} t_keyword ;

t_keyword keywords [] = { // define las palabras reservadas y los
    "main",        MAIN,           // y los token asociados
    "int",         INTEGER,
    "puts",        PUTS,
    "printf",      PRINTF,
    "&&",          AND,
    "||",          OR,
    "!=",          DIFF,
    "==",          EQUAL,
    "<=",          SMALLER,
```

```
    ">=",          BIGGER,
    "while",       WHILE,
    "for",         FOR,
    "return",      RETURN,
    "if",          IF,
    "else",        ELSE,
    NULL,          0              // para marcar el fin de la tabla
} ;

t_keyword *search_keyword (char *symbol_name)
{                              // Busca n_s en la tabla de pal. res.
                               // y devuelve puntero a registro (simbolo)
    int i ;
    t_keyword *sim ;

    i = 0 ;
    sim = keywords ;
    while (sim [i].name != NULL) {
            if (strcmp (sim [i].name, symbol_name) == 0) {
                                        // strcmp(a, b) devuelve == 0 si a==b
          return &(sim [i]) ;
        }
        i++ ;
    }

    return NULL ;
}


/*****************************************************************************/
/****************** Seccion del Analizador Lexicografico ******************/
```

```
/*****************************************************************************/

char *gen_code (char *name)     // copia el argumento a un
{                               // string en memoria dinamica
   char *p ;
   int l ;

   l = strlen (name)+1 ;
   p = (char *) my_malloc (l) ;
   strcpy (p, name) ;

   return p ;
}


int yylex ()
{
   int i ;
   unsigned char c ;
   unsigned char cc ;
   char ops_expandibles [] = "!<=>|%/&+-*" ;
   char temp_str [256] ;
   t_keyword *symbol ;

   do {
      c = getchar () ;

      if (c == '#') {       // Ignora las lineas que empiezan por #  (#define, #include)
         do {               //         OJO que puede funcionar mal si una linea contiene #
            c = getchar () ;
         } while (c != '\n') ;
```

```
    }

    if (c == '/') {          // Si la linea contiene un / puede ser inicio de comentario
        cc = getchar () ;
        if (cc != '/') {   // Si el siguiente char es /  es un comentario, pero...
            ungetc (cc, stdin) ;
        } else {
            c = getchar () ;        // ...
            if (c == '@') {          // Si es la secuencia //@  ==> transcribimos la linea
                do {                 // Se trata de codigo inline (Codigo embebido en C)
                    c = getchar () ;
                    putchar (c) ;
                } while (c != '\n') ;
            } else {                 // ==> comentario, ignorar la linea
                while (c != '\n') {
                    c = getchar () ;
                }
            }
        }
    } else if (c == '\\') c = getchar () ;

    if (c == '\n')
        n_line++ ;

} while (c == ' ' || c == '\n' || c == 10 || c == 13 || c == '\t') ;

if (c == '\"') {
    i = 0 ;
    do {
        c = getchar () ;
        temp_str [i++] = c ;
```

```
        } while (c != '\"' && i < 255) ;
        if (i == 256) {
            printf ("AVISO: string con mas de 255 caracteres en linea %d\n", n_line) ;
        }                       // habria que leer hasta el siguiente " , pero, y si falta?
        temp_str [--i] = '\0' ;
        yylval.code = gen_code (temp_str) ;
        return (STRING) ;
    }

    if (c == '.' || (c >= '0' && c <= '9')) {
        ungetc (c, stdin) ;
        scanf ("%d", &yylval.value) ;
//          printf ("\nDEV: NUMBER %d\n", yylval.value) ;        // PARA DEPURAR
        return NUMBER ;
    }

    if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z')) {
        i = 0 ;
        while (((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z') ||
            (c >= '0' && c <= '9') || c == '_') && i < 255) {
            temp_str [i++] = tolower (c) ;
            c = getchar () ;
        }
        temp_str [i] = '\0' ;
        ungetc (c, stdin) ;

        yylval.code = gen_code (temp_str) ;
        symbol = search_keyword (yylval.code) ;
        if (symbol == NULL) {    // no es palabra reservada -> identificador antes vrariabre
//              printf ("\nDEV: IDENTIF %s\n", yylval.code) ;    // PARA DEPURAR
            return (IDENTIF) ;
```

```
        } else {
//              printf ("\nDEV: OTRO %s\n", yylval.code) ;       // PARA DEPURAR
            return (symbol->token) ;
        }
    }

    if (strchr (ops_expandibles, c) != NULL) { // busca c en ops_expandibles
        cc = getchar () ;
        sprintf (temp_str, "%c%c", (char) c, (char) cc) ;
        symbol = search_keyword (temp_str) ;
        if (symbol == NULL) {
            ungetc (cc, stdin) ;
            yylval.code = NULL ;
            return (c) ;
        } else {
            yylval.code = gen_code (temp_str) ; // aunque no se use
            return (symbol->token) ;
        }
    }

//    printf ("\nDEV: LITERAL %d #%c#\n", (int) c, c) ;      // PARA DEPURAR
    if (c == EOF || c == 255 || c == 26) {
//          printf ("tEOF ") ;                              // PARA DEPURAR
        return (0) ;
    }

    return c ;
}


int main ()
```

```
{
    yyparse () ;
}
```