```
/* C sar L pez Mantec n , Paula Sub as Serrano, grupo 12 */
/* 100472092@alumnos.uc3m.es 100472119@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

typedef struct s_lista { // lista de variabbles locales
    char lista[1024][1024];
    int i;
} t_lista;

int yylex () ;
int yyerror () ;
char *mi_malloc (int) ;
char *gen_code (char *) ;
char *int_to_string (int) ;
char *char_to_string (char) ;
char *nombre_funcion;
int search_local(t_lista l, char *var);
int insert(t_lista *l, char *var);
int remove_all(t_lista *l);


char temp [2048] ;

t_lista var_list;
// 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 PRINTF        // identifica la impresion
%token PUTS          // identifica puts
%token IF
%token ELSE
%token FOR
%token RETURN


%right '='                    // minima preferencia
%left OR                      //
%left AND                     //
%left EQUAL NOTEQ             //
%left '<' LEQ '>' GEQ         //
%left '+' '-'                 //
%left '*' '/' '%'             //
%left UNARY_SIGN '!'          // maxima preferencia

%%                            // Seccion 3 Gramatica - Semantico

axioma:                                          { nombre_funcion = gen_code(""); }
            var_globales declaracion_func MAIN   { nombre_funcion = gen_code("main"); }
            '(' ')'
            '{' declaracion_variables codigo '}' { printf ("(defun main () \n%s%s)\n", $9.code, $10.code); }
            ;

var_globales : declaracion_variables             { printf ("%s", $1.code); }
```

```
            ;

declaracion_func:                                   { $$.code = ""; }
                | IDENTIF '(' args ')'
                    '{'                             { nombre_funcion = gen_code($1.code); }
                    declaracion_variables codigo
                    '}'                             {
                                                        printf("(defun %s (%s)\n%s%s)\n", $1.code, $3.code, $7.code, $8.code);
                                                        remove_all(&var_list);
                                                    }
                    declaracion_func
                ;

args:                                               { $$.code = ""; }
     | INTEGER IDENTIF rest_args                    { sprintf(temp, "%s %s", $2.code, $3.code); $$.code = gen_code (temp); }
     ;

rest_args:                                          { $$.code = ""; }
            | ',' INTEGER IDENTIF rest_args         { sprintf(temp, " %s %s", $3.code, $4.code); $$.code = gen_code(temp); }
            ;

declaracion_variables:                              { $$.code = ""; }
            | INTEGER IDENTIF
                '=' NUMBER rest_declar ';'
                declaracion_variables               {
                                                        char aux[1026] = "";
                                                        if (strcmp(nombre_funcion, "")) {
                                                            sprintf(aux, "%s_", nombre_funcion); insert(&var_list, $2.code);
                                                        }
                                                        sprintf (temp, "(setq %s%s %d) %s \n%s", aux, $2.code, $4.value, $5.code, $7.code);
                                                        $$.code = gen_code (temp);
                                                    }
            | INTEGER IDENTIF rest_declar';'
                declaracion_variables              {
                                                        char aux[1026] = "";
                                                        if (strcmp(nombre_funcion, "")) {
                                                            sprintf(aux, "%s_", nombre_funcion); insert(&var_list, $2.code);
                                                        }
                                                        sprintf(temp, "(setq %s%s 0) %s \n%s", aux, $2.code, $3.code, $5.code);
                                                        $$.code = gen_code(temp);
                                                    }
            | INTEGER IDENTIF '[' NUMBER']'
                rest_declar_vector ';'
                declaracion_variables               {
                                                        char aux[1026] = "";
                                                        if (strcmp(nombre_funcion, "")) {
                                                            sprintf(aux, "%s_", nombre_funcion); insert(&var_list, $2.code);
                                                        }
                                                        sprintf(temp, "(setq %s%s (make-array %d)) %s \n%s",aux, $2.code, $4.value, $6.code, $8.code);
                                                        $$.code = gen_code(temp);
                                                    }
            ;

rest_declar_vector:     /* lambda */                { $$.code = "" ; }
                        | ','
                        IDENTIF '[' NUMBER']'
                        rest_declar_vector          {
                                                        char aux[1026] = "";
                                                        if (strcmp(nombre_funcion, "")) {
                                                            sprintf(aux, "%s_", nombre_funcion); insert(&var_list, $2.code);
                                                        }
                                                        sprintf(temp, "(setq %s%s (make-array %d)) %s",aux, $2.code, $4.value, $6.code);
                                                        $$.code = gen_code(temp);
                                                    }
            ;

rest_declar:    /* lambda */                        { $$.code = "" ; }
                | ','
                    IDENTIF '=' NUMBER
                    rest_declar                     {
                                                        char aux[1026] = "";
                                                        if (strcmp(nombre_funcion, "")) {
                                                            sprintf(aux, "%s_", nombre_funcion); insert(&var_list, $2.code);
                                                        }
                                                        sprintf (temp, "(setq %s%s %d) %s", aux,$2.code, $4.value, $5.code) ;
```

```
                                                $$.code = gen_code (temp);
                                            }
                | ',' IDENTIF  rest_declar      {
                                                char aux[1026] = "";
                                                if (strcmp(nombre_funcion, "")) {
                                                    sprintf(aux, "%s_", nombre_funcion); insert(&var_list, $2.code);
                                                }
                                                sprintf (temp, "(setq %s%s 0) %s", aux,$2.code, $3.code) ;
                                                $$.code = gen_code (temp) ;
                                            }
                ;


codigo:     sentencia ';' r_expr                { sprintf (temp, "%s\n%s", $1.code, $3.code); $$.code = gen_code (temp); }
            | WHILE '(' expresion ')'
                '{' codigo '}'
                r_expr                          { sprintf(temp, "(loop while %s do\n%s)\n%s", $3.code, $6.code, $8.code); $$.code = gen_code(temp); }
            | IF '(' expresion ')'
                '{' codigo '}'
                est_else
                r_expr                          { sprintf(temp, "(if %s\n(progn %s)\n%s)\n%s", $3.code, $6.code, $8.code, $9.code); $$.code = gen_code(
    temp); }
            | FOR
                '('
                inicializar ';'
                expresion ';'
                IDENTIF '=' incr_decr
                ')'
                '{' codigo '}' r_expr           {
                                                char aux[1026] = "";
                                                if (search_local(var_list, $7.code)) {
                                                    sprintf(aux, "%s_", nombre_funcion);
                                                }
                                                sprintf(temp, "%s(loop while %s do\n%s(setf %s%s%s)) \n%s", $3.code, $5.code, $12.code, aux, $7.code,
    $9.code, $14.code);
                                                $$.code = gen_code(temp);
                                            }
            ;

inicializar: IDENTIF '=' expresion              {
                                                char aux[1026] = "";
                                                if (search_local(var_list, $1.code)) {
                                                    sprintf(aux, "%s_", nombre_funcion);
                                                }
                                                sprintf (temp, "(setf %s%s %s)", aux,$1.code, $3.code) ;
                                                $$.code = gen_code (temp) ;
                                            }
            ;

incr_decr:    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); }

est_else:                                       { $$.code = ""; }
            | ELSE '{' codigo '}'               { sprintf(temp, "(progn %s)\n", $3.code); $$.code = gen_code(temp); }
            ;

r_expr:                                 { $$.code = ""; }
            |   codigo                  { $$ = $1; }
            ;

sentencia:    IDENTIF '=' expresion     {
                                        char aux[1026] = "";
                                        if (search_local(var_list, $1.code)) {
                                            sprintf(aux, "%s_", nombre_funcion);
                                        }
                                        sprintf (temp, "(setf %s%s %s)", aux,$1.code, $3.code) ;
                                        $$.code = gen_code (temp);
                                    }
            | vector '=' expresion  {
                                        char aux[1026] = "";
                                        if (search_local(var_list, $1.code)) {
                                            sprintf(aux, "%s_", nombre_funcion);
                                        }
                                        sprintf (temp, "(setf %s%s %s)",aux, $1.code, $3.code) ;
                                        $$.code = gen_code (temp);
```

```
                                    }
            | RETURN expresion                                  {sprintf(temp, "(return-from %s %s)", nombre_funcion,$2.code);
                                                                $$.code = gen_code (temp) ;}
            | PRINTF
                '('
                STRING ','
                expresion rest_print
                ')'                 {
                                        sprintf (temp, "(prin1 %s) %s", $5.code, $6.code) ;
                                        $$.code = gen_code (temp) ;
                                    }
            | PRINTF
                '('
                STRING ','
                STRING rest_print
                ')'                 {
                                        sprintf (temp, "(prin1 \"%s\") %s", $5.code, $6.code) ;
                                        $$.code = gen_code (temp);
                                    }
            | PUTS
                '(' STRING ')'      {
                                        sprintf (temp, "(print \"%s\")", $3.code) ;
                                        $$.code = gen_code (temp);
                                    }
            | IDENTIF
                '('
                expresion
                rest_params
                ')'                 {
                                        sprintf (temp, "(%s %s %s)", $1.code,$3.code, $4.code) ;
                                        $$.code = gen_code (temp);
                                    }
            | IDENTIF '(')'         { sprintf (temp, "(%s)", $1.code); $$.code = gen_code (temp); }
            ;

rest_print:                         { $$.code = "" ; }
            | ',' expresion rest_print  { sprintf(temp, "(prin1 %s) %s", $2.code, $3.code); $$.code = gen_code(temp); }
            | ',' STRING rest_print     { sprintf(temp, "(prin1 \"%s\") %s", $2.code, $3.code); $$.code = gen_code(temp); }
            ;

rest_params:                            { $$.code = ""; }
            | ',' expresion rest_params     {
                                               sprintf(temp, "%s %s", $2.code, $3.code);
                                               $$.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); $$.code = gen_code (temp); }
            |   expresion '%' expresion     { sprintf (temp, "(mod %s %s)", $1.code, $3.code); $$.code = gen_code (temp); }
            |   expresion AND expresion     { sprintf (temp, "(and %s %s)", $1.code, $3.code); $$.code = gen_code (temp); }
            |   expresion OR expresion      { sprintf (temp, "(or %s %s)", $1.code, $3.code); $$.code = gen_code (temp); }
            |   expresion NOTEQ 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 GEQ expresion     { sprintf (temp, "(>= %s %s)", $1.code, $3.code); $$.code = gen_code (temp); }
            |   expresion LEQ expresion     { sprintf (temp, "(<= %s %s)", $1.code, $3.code); $$.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); }
            | IDENTIF
                '('expresion rest_params')'     { sprintf (temp, "(%s %s %s)", $1.code,$3.code, $4.code); $$.code = gen_code (temp); }
            | IDENTIF '(')'                     { sprintf (temp, "(%s)", $1.code); $$.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) ;
                                                  $$.code = gen_code (temp) ; }
            |   '!' operando %prec UNARY_SIGN   { sprintf(temp, "(not %s)", $2.code);
                                                  $$.code = gen_code(temp);}
            ;
```

```
operando:       IDENTIF                 {
                                            char aux[1026] = "";
                                            if (search_local(var_list, $1.code)) {
                                                    sprintf(aux, "%s_", nombre_funcion);
                                                }
                                            sprintf (temp, "%s%s", aux,$1.code) ;
                                            $$.code = gen_code (temp);
                                        }
            |   NUMBER                  {
                                            sprintf (temp, "%d", $1.value);
                                            $$.code = gen_code (temp);
                                        }
            |   '(' expresion ')'       { $$ = $2 ; }
            | vector                    { sprintf (temp, "%s", $1.code); $$.code = gen_code (temp); }
            ;

vector:      IDENTIF '[' expresion ']'     {
                                                char aux[1026] = "";
                                                if (search_local(var_list, $1.code)) {
                                                    sprintf(aux, "%s_", nombre_funcion);
                                                }
                                                sprintf (temp, "(aref %s%s %s)", aux, $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 ;
}
/**
* Busca una variable en la lista de locales.
*
*/
int search_local(t_lista l, char *var) {
    for (int i = 0; i < l.i; ++i) {
        if (strcmp(l.lista[i], var) == 0) { // se encuentra la variable
            return 1;
        }
```

```
376|     }
377|     return 0;
378| }
379|
380| /**
381| * inserta una variable local en la lista
382| *
383| */
384| int insert(t_lista *l, char *var) {
385|     strcpy(&(l->lista[l->i][0]), var);
386|
387|     ++(l->i);
388|     return l->i;
389| }
390|
391| /**
392| * vac a la lista de variables locales
393| *
394| */
395| int remove_all(t_lista *l) {
396|     l->i = 0;
397|     return 0;
398| }
399|
400| /*****************************************************************************/
401| /********************* Seccion de Palabras Reservadas ********************/
402| /*****************************************************************************/
403|
404| typedef struct s_keyword { // para las palabras reservadas de C
405|     char *name ;
406|     int token ;
407| } t_keyword ;
408|
409| t_keyword keywords [] = { // define las palabras reservadas y los
410|     "main",         MAIN,           // y los token asociados
411|     "int",          INTEGER,
412|     "puts",         PUTS,
413|     "printf",       PRINTF,
414|     "&&",           AND,
415|     "||",           OR,
416|     "!=",           NOTEQ,
417|     "==",           EQUAL,
418|     "<=",           LEQ,
419|     ">=",           GEQ,
420|     "while",        WHILE,
421|     "if",           IF,
422|     "else",         ELSE,
423|     "for",          FOR,
424|     "return",        RETURN,
425|     NULL,           0               // para marcar el fin de la tabla
426| } ;
427|
428| t_keyword *search_keyword (char *symbol_name)
429| {                                   // Busca n_s en la tabla de pal. res.
430|                                     // y devuelve puntero a registro (simbolo)
431|     int i ;
432|     t_keyword *sim ;
433|
434|     i = 0 ;
435|     sim = keywords ;
436|     while (sim [i].name != NULL) {
437|       if (strcmp (sim [i].name, symbol_name) == 0) {
438|                                     // strcmp(a, b) devuelve == 0 si a==b
439|             return &(sim [i]) ;
440|         }
441|         i++ ;
442|     }
443|
444|     return NULL ;
445| }
446|
447|
448| /*****************************************************************************/
449| /******************* Seccion del Analizador Lexicografico ********************/
450| /*****************************************************************************/
451|
452| 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 () ;
```

Listing 1: trad.y