Implementación en C de traductores descendentes

[versión 1.0, 2006-02-20, JosuKa]

1. Infija a postfija con descendente recursivo

1a. Para obtener el compilador

```
rem cll1.bat (c) JosuKa Díaz Labrador 2006
flex -oinfijalex.c infija.lex
gcc infijalex.c infijamain.c -o inf2post.exe
```

1b. Declaraciones

```
/* infija.h (versión final ANT)
   Traductor de expresiones en notación infija a notación postfija
   (c) JosuKa Díaz Labrador 2006
*/

/* el tipo token: tiene que ser el mismo de los defines de más abajo */

typedef int TTOKEN;

/* el tipo atributo: hay dos posibles tipos */

typedef union {
        int  TCodigo;
        char *TCadena;
} YYSTYPE;

/* declaración para el programa lex */

extern YYSTYPE atribANT;

/* lista de nombres de tokens */

#define DOLAR    0       /* 0 porque así lo quiere lex */

#define PAR_ABR 258
#define PAR_CER 259
#define PTO_COMA 260
#define NUM     261
#define OP_ADIT 262
#define OP_MULT 263

extern void yyerror( char * );
```

1c. Analizador léxico en lex

```
%{
/* infija.lex (versión final ANT)
   Traductor de expresiones en notación infija a notación postfija
   Analizador léxico
   (c) JosuKa Díaz Labrador 2006
*/

#include <string.h>
#include "infija.h"

void yyerror( char *msg )
{
        printf( "\nError en linea %d con '%s': %s", yylineno, yytext, msg );
}

%}

%option noyywrap
%option yylineno
%option never-interactive

/* Definiciones Regulares */

digito          [0-9]
constante       {digito}+
sep             [ \t\n]

%%

";"             { atribANT.TCodigo = 0; return( PTO_COMA ); }

"+"             { atribANT.TCodigo = 1; return( OP_ADIT ); }
"-"             { atribANT.TCodigo = 2; return( OP_ADIT ); }

"*"             { atribANT.TCodigo = 1; return( OP_MULT ); }
"/"             { atribANT.TCodigo = 2; return( OP_MULT ); }
"%"             { atribANT.TCodigo = 3; return( OP_MULT ); }

"("             { atribANT.TCodigo = 0; return( PAR_ABR ); }
")"             { atribANT.TCodigo = 0; return( PAR_CER ); }

{constante}     { atribANT.TCadena = strdup( yytext ); return( NUM ); }

{sep}           { }

.               { yyerror( "lexical error: caracter desconocido" ); }

%%
```

1d. Analizador sintáctico descendente recursivo

```
/* infijamain.c (versión final ANT)
   Traductor de expresiones en notación infija a notación postfija
   Analizador sintáctico descendente recursivo
   (c) JosuKa Díaz Labrador 2006
*/

#include "infija.h"

TTOKEN tokenACT, tokenANT;
YYSTYPE atribACT, atribANT;

void parear( TTOKEN token )
{
        if( token == tokenANT ) {
                tokenACT = tokenANT; atribACT = atribANT;
                tokenANT = yylex();
                        /* esta llamada actualiza lateralmente atribANT */
        } else
                yyerror( "syntactic error: token incorrecto" );
}

/* Los procedimientos de los no-terminales */

extern void S();
extern void L();
extern void E();
extern void Ep();
extern void T();
extern void Tp();
extern void F();

void S()
{
        switch( tokenANT ) {
        case NUM:
        case PAR_ABR:
        case DOLAR:
                L();
                break;
        default:
                yyerror( "syntactic error: regla incorrecta" );
                break;
        }
}

void L()
{
        switch( tokenANT ) {
        case NUM:
        case PAR_ABR:
                E();
                parear(PTO_COMA);

                printf( "\n" );

                L();
                break;
        case DOLAR:
                /* regla vacía */
                break;
```

```
        default:
                yyerror( "syntactic error: regla incorrecta" );
                break;
        }
}

void E()
{
        switch( tokenANT ) {
        case NUM:
        case PAR_ABR:
                T();
                Ep();
                break;
        default:
                yyerror( "syntactic error: regla incorrecta" );
                break;
        }
}

void Ep()
{
        switch( tokenANT ) {
        case OP_ADIT:
                parear(OP_ADIT);
                T();
                Ep();
                break;
        case PTO_COMA:
        case PAR_CER:
                /* regla vacía */
                break;
        default:
                yyerror( "syntactic error: regla incorrecta" );
                break;
        }
}

void T()
{
        switch( tokenANT ) {
        case NUM:
        case PAR_ABR:
                F();
                Tp();
                break;
        default:
                yyerror( "syntactic error: regla incorrecta" );
                break;
        }
}

void Tp()
{
        switch( tokenANT ) {
        case OP_MULT:
                parear(OP_MULT);
                F();
                Tp();
                break;
        case OP_ADIT:
        case PTO_COMA:
        case PAR_CER:
```

```
                /* regla vacía */
                break;
        default:
                yyerror( "syntactic error: regla incorrecta" );
                break;
        }
}

void F()
{
        switch( tokenANT ) {
        case NUM:
                parear(NUM);

                printf( "%s ", atribACT.TCadena );

                break;
        case PAR_ABR:
                parear(PAR_ABR);
                E();
                parear(PAR_CER);
                break;
        default:
                yyerror( "syntactic error: regla incorrecta" );
                break;
        }
}

main( int argc, char *argv[] )
{
        tokenACT = atribACT.TCodigo = 0;
        tokenANT = yylex();
                /* esta llamada actualiza lateralmente atribANT */
        S();
        if( tokenANT != DOLAR )
                yyerror( "syntactic error: programa debería haber acabado" );
}
```

2. Infija a postfija con descendente iterativo

2a. Para obtener el compilador (nuevo módulo de pila)

```
rem cll1.bat (c) JosuKa Díaz Labrador 2006
flex -oinfijalex.c infija.lex
gcc infijalex.c tpila.c infijamain.c -o inf2post.exe
```

2b. Declaraciones (ligeramente distinto del recursivo)

```
/* infija.h (versión final ANT)
   Traductor de expresiones en notación infija a notación postfija
   (c) JosuKa Díaz Labrador 2006
*/

/* el tipo token: al usar lex, debe ser int */

typedef int TTOKEN;

/* el tipo atributo: hay dos posibles tipos */

typedef union {
        int  TCodigo;
        char *TCadena;
} YYSTYPE;

/* declaración para el programa lex */

extern YYSTYPE atribANT;

/* lista de nombres de tokens */

#define DOLAR   0       /* 0 porque así lo quiere lex */

#define SDOLAR  257     /* falso DOLAR: mejor el primero */
#define PAR_ABR 258
#define PAR_CER 259
#define P_COMA  260
#define NUM     261
#define OP_ADIT 262
#define OP_MULT 263

extern void yyerror( char * );
```

2c. Analizador léxico en lex

Es exactamente el mismo que el listado en el apartado 1c.

2d. Manejo de pilas (declaraciones)

```
/* tpila.h
   TAD (bueno, quizás no tanto) pila
   (c) JosuKa Díaz Labrador 2006
*/

#include "infija.h"

/* la pila puede tener los mismos tipos que los atributos: YYSTYPE */

struct tpila {
        YYSTYPE       content;
        struct tpila *sig;
};
```

```
typedef struct tpila TPILA;

TPILA  * pilaNOVACIA( TPILA *pila );
TPILA  * pilaPUSH( TPILA *pila, YYSTYPE data );
YYSTYPE  pilaTOP( TPILA *pila );
TPILA  * pilaPOP( TPILA *pila );
```

2e. Manejo de pilas (módulo)

```
/* tpila.c
   TAD (bueno, quizás no tanto) pila
   (c) JosuKa Díaz Labrador 2006
*/

#include <stdlib.h>
#include "tpila.h"

TPILA *pilaNOVACIA( TPILA *pila )
{
        return( pila );          /* pilaNOVACIA(pila) = pila */
}

TPILA *pilaPUSH( TPILA *pila, YYSTYPE data )
{
        TPILA *newtop;

        newtop = (TPILA *)calloc( 1, sizeof( TPILA ) );
        newtop->sig = pila;
        newtop->content = data;
        return( newtop );
}

YYSTYPE pilaTOP( TPILA *pila )
{
        if( ! pila ) {
                printf( "\n¡¡CHAVAL!!\n" );
                exit( 2 );
        }
        return( pila->content );
}

TPILA *pilaPOP( TPILA *pila )
{
        TPILA *newtop;

        if( ! pila ) {
                printf( "\n¡¡CHAVAL!!\n" );
                exit( 3 );
        }
        newtop = pila->sig;
        free( pila );
        return( newtop );
}
```

2f. Analizador sintáctico descendente iterativo

```
/* infijamain.c (versión iterativa ANT)
   Traductor de expresiones en notación infija a notación postfija
   Analizador sintáctico descendente iterativo
   (c) JosuKa Díaz Labrador 2006
*/

#include "tpila.h"

/* ya incluye infija.h */

/* Terminales de la gramática: están en infija.h */

#define TER_OFFSET 257

/* orden correlativo de códigos:
   SDOLAR PAR_ABR PAR_CER P_COMA NUM OP_ADIT OP_MULT
   recordar que DOLAR debe transformarse en SDOLAR
   con la siguiente función
*/

TTOKEN falsoyylex()
{
        TTOKEN token;

        token = yylex();
        if( token == DOLAR ) token = SDOLAR;
        return( token );
}

/* Variables de la gramática */

#define VAR_OFFSET 1000

#define VAR_S    1000
#define VAR_L    1001
#define VAR_E    1002
#define VAR_Ep   1003
#define VAR_T    1004
#define VAR_Tp   1005
#define VAR_F    1006

/* Acciones semánticas */

#define ACC_OFFSET 2000

#define ACC_A1   2000
#define ACC_A2   2001
#define ACC_A3   2002
#define ACC_A4   2003
#define ACC_A5   2004
#define ACC_A6   2005

/* Reglas de la gramática; por orden desde 0:
        0       regla de relleno (todo 0)
        1       S  -> L
        2       L  -> lambda
        3       L  -> E pto_coma A1 L
        4       E  -> T Ep
        5       Ep -> lambda
        6       Ep -> op_adit A2 T A3 Ep
        7       T  -> F Tp
```

```
        8       Tp -> lambda
        9       Tp -> op_mult A4 F A5 Tp
        10      F  -> n A6
        11      F  -> ( E )
   trucos:
   * solo hacen falta partes derechas
   * se meten ya invertidas
*/

int reglasG[12][6] = {
/*  0 */  { 0,       0,      0,       0,      0,       0 }
/*  1 */ ,{ VAR_L,   0,      0,       0,      0,       0 }
/*  2 */ ,{ 0,       0,      0,       0,      0,       0 }
/*  3 */ ,{ VAR_L,   ACC_A1, P_COMA,  VAR_E,  0,       0 }
/*  4 */ ,{ VAR_Ep,  VAR_T,  0,       0,      0,       0 }
/*  5 */ ,{ 0,       0,      0,       0,      0,       0 }
/*  6 */ ,{ VAR_Ep,  ACC_A3, VAR_T,   ACC_A2, OP_ADIT, 0 }
/*  7 */ ,{ VAR_Tp,  VAR_F,  0,       0,      0,       0 }
/*  8 */ ,{ 0,       0,      0,       0,      0,       0 }
/*  9 */ ,{ VAR_Tp,  ACC_A5, VAR_F,   ACC_A4, OP_MULT, 0 }
/* 10 */ ,{ ACC_A6,  NUM,    0,       0,      0,       0 }
/* 11 */ ,{ PAR_CER, VAR_E,  PAR_ABR, 0,      0,       0 }
};

/* Tabla LL(1) */

int tablaLL1[7][7] = {
/* cols:     SDOLAR PAR_ABR PAR_CER P_COMA NUM OP_ADIT OP_MULT */
/* S  */  {  1,     1,      0,      0,     1,  0,      0       }
/* L  */ ,{  2,     3,      0,      0,     3,  0,      0       }
/* E  */ ,{  0,     4,      0,      0,     4,  0,      0       }
/* Ep */ ,{  0,     0,      5,      5,     0,  6,      0       }
/* T  */ ,{  0,     7,      0,      0,     7,  0,      0       }
/* Tp */ ,{  0,     0,      8,      8,     0,  8,      9       }
/* F  */ ,{  0,    11,      0,      0,    10,  0,      0       }
};

/* Las variables globales */

TTOKEN  tokenACT, tokenANT;
YYSTYPE atribACT, atribANT;

void parear( TTOKEN token )
{
        if( token == tokenANT ) {
                tokenACT = tokenANT; atribACT = atribANT;
                tokenANT = falsoyylex();
                        /* esta llamada actualiza lateralmente atribANT */
        } else
                yyerror( "syntactic error: token incorrecto" );
}

TPILA *pilaSint;
TPILA *pilaSem;

/* Las acciones semánticas */

void accA1()
{
        printf( "\n" );
}

void accA2() {}
```

```
void accA3() {}

void accA4() {}

void accA5() {}

void accA6()
{
        printf( "%s ", atribACT.TCadena );
}

/* El índice de acciones: esta sí que es buena */

void (*tablaACC[6])() = {
        accA1, accA2, accA3, accA4, accA5, accA6
};

main( int argc, char *argv[] )
{
        YYSTYPE toppila;
        int regla, var, term;
        int i;

        tokenACT = atribACT.TCodigo = 0;
        tokenANT = falsoyylex();
                /* esta llamada actualiza lateralmente atribANT */

        toppila.TCodigo = VAR_S;
        pilaSint = pilaPUSH( pilaSint, toppila );

                /* empieza el tema */
        while( pilaNOVACIA( pilaSint ) ) {
                toppila = pilaTOP( pilaSint );
                pilaSint = pilaPOP( pilaSint );
                if( toppila.TCodigo >= ACC_OFFSET ) {
                                /* es una accion: ejecutar */
                        tablaACC[ toppila.TCodigo - ACC_OFFSET ]();
                } else if( toppila.TCodigo >= VAR_OFFSET ) {
                                /* es una variable: expandir */
                        var = toppila.TCodigo - VAR_OFFSET;
                        term = tokenANT - TER_OFFSET;
                        regla = tablaLL1[var][term];
                        if( regla == 0 )
                                yyerror( "syntactic error: no hay regla" );
                        else
                                for( i=0; reglasG[regla][i]; i++ ) {
                                        toppila.TCodigo = reglasG[regla][i];
                                        pilaSint = pilaPUSH( pilaSint, toppila );
                                }
                } else /* se supone que solo puede ser terminal: parear */
                        parear( toppila.TCodigo );
        }

                /* acabó el tema */
        if( tokenANT != SDOLAR )
                yyerror( "syntactic error: programa debería haber acabado" );
}
```