UNIVERSIDAD AUTÓNOMA DE CHIHUAHUA

TEORÍA DE LA COMPUTACIÓN

Programación de un Intérprete pt. 2

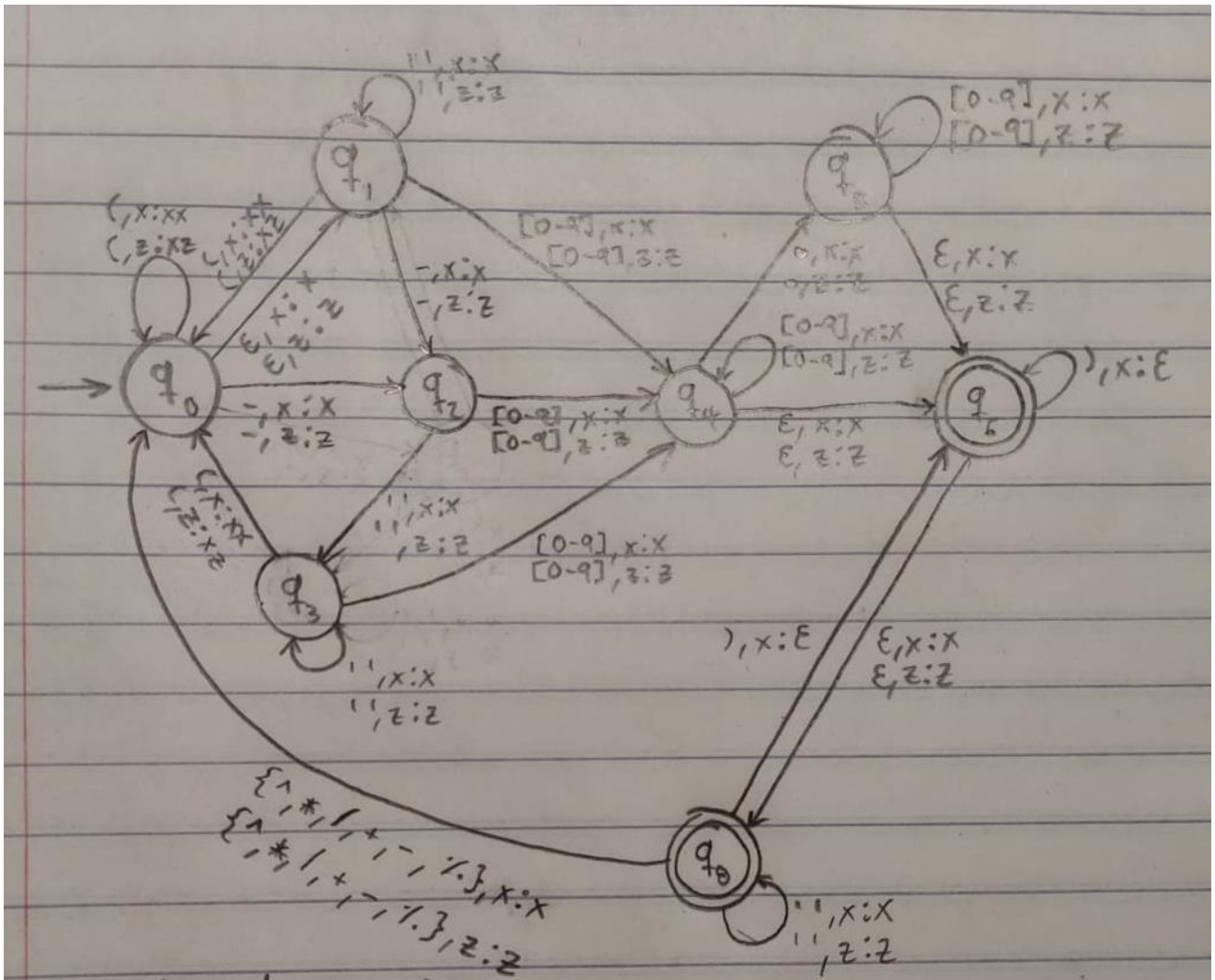
Alumnos / Matriculas:

- Erick Fernando Nevarez Ávila / 357664
- Adrian Alejandro Gonzales Dominguez / 359834
- Héctor Daniel Medrano Meza / 361345

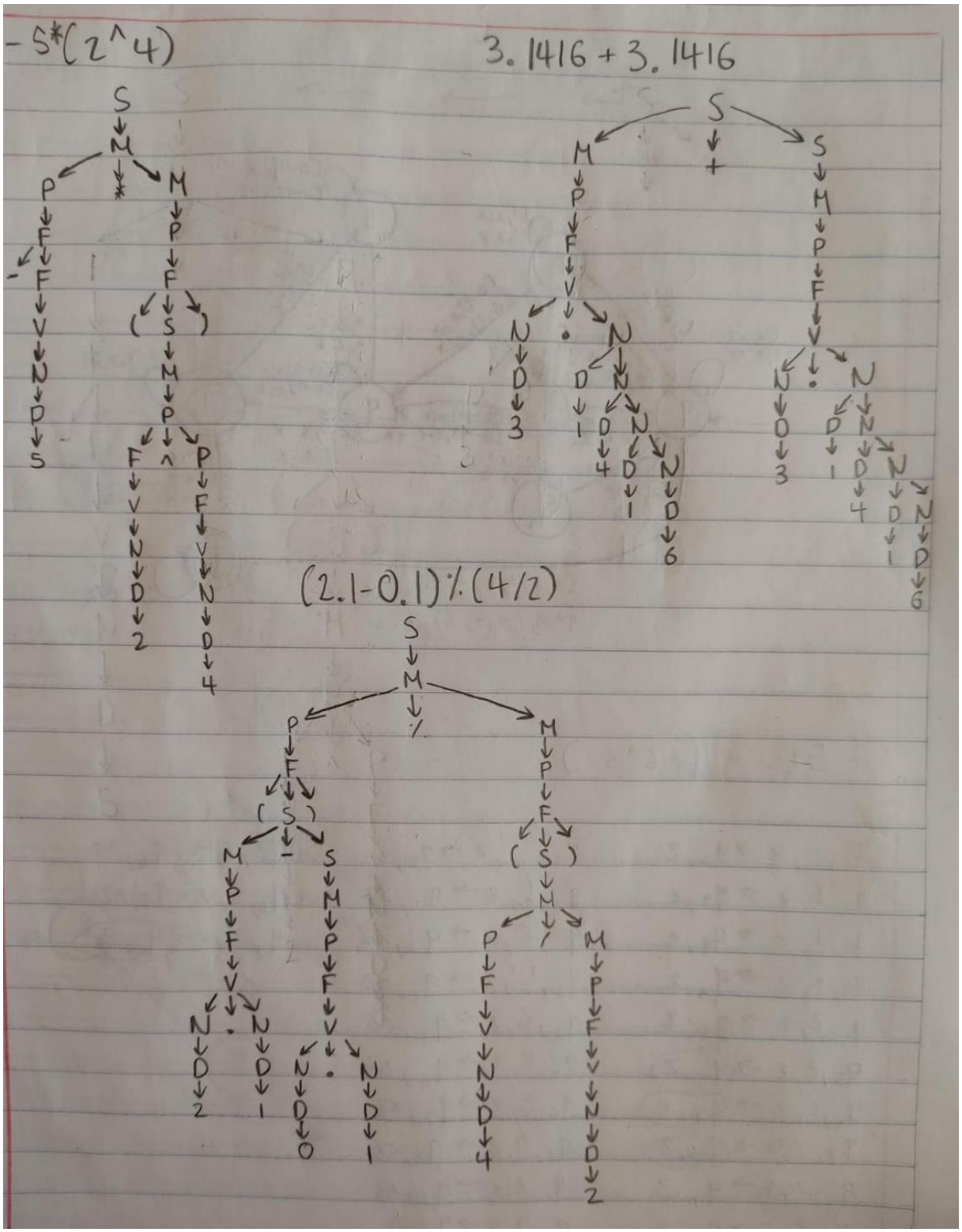
Docente: M.I. Mario Andrés Cuevas Gutierrez

Fecha: 2024-10-13

GRAMÁTICA + AUTÓMATA



ÁRBOLES DE DERIVACIÓN



PRUEBAS DE FUNCIONAMIENTO

```
3.136 + 3.1416
6.28
OK
-5          *           (2    ^    4  )
-80.00
OK
(2.1 - 0.1)%       (4 /2)
0.00
OK
```

CÓDIGO

lexer.c

```
#include "lexer.h"

int edos[][13] = {
    { 1,  2,  3,  4,  5,  6,  13, 14, 8,  12, 7,  0,  11},
    { 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11},
    { 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11},
    { 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11},
    { 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11},
    { 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11},
    { 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11},
    { 10, 10, 10, 10, 10, 10, 10, 10, 8,  10, 7,  10, 10},
    { 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 9,  11, 11},
    { 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 9,  10, 10},
    { 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11},
    { 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11},
    { 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11},
    { 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11},
    { 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11}
    };

char alfabeto[] = {'(', ')', '+', '-', '*', '/', '%', '^', '.', '\n'};
const int c_alfabeto = 10;

token_t get_token (FILE *entrada);
void unget_token (token_t t, FILE *entrada);
int elegir_columna (char c);
```

```
token_t get_token (FILE *entrada){
    int edo = 0;
    char c;
    token_t t;
    t.lex_len = 0;

    while (!feof (entrada)) {
        c = getc (entrada);
        t.lexeme [t.lex_len++] = c;
        t.lexeme [t.lex_len] = '\0';
        int columna = elegir_columna (c);
        edo = edos [edo][columna];

        switch (edo) {
            case 1:
                t.type = L_P;
                return t;
            case 2:
                t.type = R_P;
                return t;
            case 3:
                t.type = ADD;
                return t;
            case 4:
                t.type = SUB:
                return t;
            case 5:
                t.type = MULT;
                return t;
```

```
        case 6:
            t.type = DIV;
            return t;
        case 10:
            ungetc(c, entrada);
            t.lexeme[--t.lex_len] = '\0';
            t.type = NUM;
            return t;
        case 11:
            t.type = ERR;
            return t;
        case 12:
            t.type = ENDL;
            return t;
        case 13:
            t.type = MOD;
            return t;
        case 14:
            t.type = POT;
            return t;
        }
    }

    t.lexeme[0] = '\0';
    t.lex_len = 0;
    t.type = ENDF;
    return t;
}
```

```
void unget_token (token_t t, FILE *entrada) {
        for (int i = t.lex_len -1; i>=0; i--) {
                ungetc (t.lexeme [i], entrada);
        }
}

int elegir_columna (char c) {
        int i = 0;
        while (i < c_alfabeto) {
                if (alfabeto [i] == c) return i;
                i++;
        }
        if (isdigit (c)) return i;
        i++;
        if (isblank (c)) return i;
        i++;
        return i;
}
```

lexer.h

```
#ifndef _LEXER_
#define _LEXER_

#include <stdio.h>
#include <ctype.h>

#define LEX_LEN 100
enum lex_type {
        L_P, R_P, ADD, SUB, MULT, DIV, NUM, ERR, ENDL, ENDF,
        MOD, POT
};

struct token{
        char lexeme[LEX_LEN];
        int lex_len;
        enum lex_type type;
};

typedef struct token token_t;

extern token_t get_token (FILE *entrada);
extern void unget_token (token_t t, FILE *entrada);

#endif
```

parser.c

```c
#include <stdio.h>
#include "stack.h"
#include <ctype.h>
#include "lexer.h"
#include <math.h>

FILE *entrada;

Snode_t *pilha = NULL;

int S();
int E();
int M();
int P();
int F();
int N();

int main(int argc, char **args){
  if(argc==2) entrada = fopen(args[1], "r");
  else entrada = stdin;
  while(!feof(entrada)){
    if(S()){
      double *resultado = pop(&pilha);
      printf("%.2lf\n", *resultado);
      printf("OK");
    }
    else while(!feof(entrada) && (getc(entrada)!='\n'));
    putc('\n', stdout);
  }
  return 0;
}
```

```
int S(){
    if(E()){
        token_t t = get_token(entrada);
        if(t.type == ENDL){
            return 1;
        }
    }
}

int E(){
    if(M()){
        token_t t = get_token(entrada);
        if(t.type == ADD){
            if(E()){
                double *b = pop(&pila);
                double *a = pop(&pila);
                double *c = malloc(sizeof(double));
                *c = *a + *b;
                push(&pila, c);
                free(a); free(b);
                return 1;
            }
            fprintf(stderr, "Se esperaba algo despues de '*'\n");
            return 0;
        }
        if(t.type == SUB){
            if(E()){
                double *b = pop(&pila);
                double *a = pop(&pila);
                double *c = malloc(sizeof(double));
                *c = *a - *b;
                push(&pila, c);
                free(a); free(b);
```

```
            return 1;
        }
        fprintf(stderr, "Se esperaba algo despues de '\n'");
        return 0;
    }
    ungetc(t, entrada);
    return 1;
  }

  return 0;
}

int M(){
  if(P()){
    token_t t = get_token(entrada);
    if(t.type == MULT){
      if(M()){
        double *b = pop(&pila);
        double *a = pop(&pila);
        double *c = malloc(sizeof(double));
        *c = *a * *b;
        push(&pila, c);
        free(a); free(b);
        return 1;
      }
      fprintf(stderr, "Se esperaba algo despues de '*'\n");
      return 0;
    }
    if(t.type == DIV){
      if(M()){
        double *b = pop(&pila);
        double *a = pop(&pila);
        double *c = malloc(sizeof(double));
        *c = *a / *b;
```

```
            push(&pila,c);
            free(a); free(b);
            return 1;
        }
        fprintf(stderr, "Se esperaba algo despues de '^'\n");
        return 0;
    }
    if(t.type == MOD){
        if(M()){
            double *b = pop(&pila);
            double *a = pop(&pila);
            double *c = malloc(sizeof(double));
            *c = fmod(*a, *b);
            push(&pila,c);
            free(a); free(b);
            return 1;
        }
        fprintf(stderr, "Se esperaba algo despues de '%'\n");
        return 0;
    }
    ungetToken(t,entrada);
    return 1;
}
return 0;
}

int P(){
    if(F()){
        Token t = getToken(entrada);
        if(t.type == POT){
            if(P()){
                double *b = pop(&pila);
                double *a = pop(&pila);
```

```
                double *c = malloc(sizeof(double));
                *c = pow(*a, *b);
                push(&pila, c);
                free(a); free(b);
                return 1;
            }
            fprintf(stderr, "Se esperaba algo despues de '^'\n");
            return 0;
        }
        ungetToken(t, entrada);
        return 1;
    }
    return 0;
}

int F(){
    if(N()) return 1;
    Token t = getToken(entrada);
    if(t.type == SUB){
        if(F()){
            double *valor = top(&pila);
            *valor *= -1;
            return 1;
        }
        fprintf(stderr, "Se esperaba un factor despues de '-'\n");
        return 0;
    }
    if(t.type == L_P){
        if(E()){
            Token t = getToken(entrada);
            if(t.type == R_P){
                return 1;
            }
```

```
            ungetToken(t, entrada);
            fprintf(stderr, "Se esperaba un ')'\n");
            return 0;
        }
    }
    ungetToken(t, entrada);
    return 0;
}

int N(){
    token_t t = getToken(entrada);
    if(t.type == NUM){
        double *valor = malloc(sizeof(
        *valor = atof(t.lexeme);
        push(&pila, valor);
        return 1;
    }
    ungetToken(t, entrada);
    return 0;
}
```