Instituto Politécnico Nacional.

Escuela Superior De Cómputo.

Materia:

Compiladores.

Tema:

Reporte.

(Práctica 1).

Profesor:

Tecla Parra Roberto.

Alumno:

Mario Alberto Miranda Sandoval.

Grupo:

3CM7

Fecha:

24 - Febrero - 2020.

Objetivo de la práctica.

Suponga que cuenta con el código del producto punto, el producto punto cruz, la multiplicación por un escalar, la suma, la resta y la magnitud. Escribir una especificación de yacc para evaluar expresiones que involucren operaciones con vectores.

Entrada

[1 2 3] + [2 4 6]

Salida

[3 6 9]

Desarrollo.

- YACC
- LEX
- Visual Studio Code

```
%union{
    double val;
    Vector *vector;
}

%token<val> NUMBER

%type<vector> exp
%type<vector> vector
%type<val> number
```

Primero definimos nuestros tokens de acuerdo a la unión definida con anterioridad.

```
%left '+' '-'
%left '*'
%left 'x' '.'
```

Ahora definimos la precedencia y asociatividad de los operadores para eliminar la ambigüedad.

```
%%
    inputString:
      | inputString list;
      ;

    list: '\n'
```

```
7.        | exp     '\n'  {imprimeVector($1);}
8.        | number  '\n'  {printf("%lf\n", $1);}
9.        ;
10.
11.     exp: vector
12.       | exp '+' exp     {$$ = sumaVector($1, $3);}
13.       | exp '-' exp     {$$ = restaVector($1, $3);}
14.       | exp '*' NUMBER     {$$ = escalarVector($3, $1);}
15.       | NUMBER '*' exp  {$$ = escalarVector($1, $3);}
16.       | exp 'x' exp     {$$ = productoCruz($1, $3);}
17.     ;
18.
19.     number: NUMBER
20.       | vector '.' vector {$$ = productoPunto($1, $3);}
21.       | '|' vector '|' {$$ = vectorMagnitud($2);}
22.       ;
23.
24.     vector : '[' NUMBER NUMBER NUMBER ']' {Vector *v = creaVector(3);
25.                                           v -> vec[0] = $2;
26.                                           v -> vec[1] = $3;
27.                                           v -> vec[2] = $4;
28.                                           $$ = v;}
29.       ;
30. %%
```

Ahora definimos la sección de reglas, haciendo énfasis en donde creamos el vector, observamos que especificamos que vamos a recibir entre corchetes 3 tokens del tipo number (double), llamamos a la función **crearVector** y recibiremos un vector como "respuesta", de manera análoga creamos las reglas para las operaciones, como bien sabemos las únicas operaciones que al efectuarse nos regresan un valor escalar son, el producto punto y la magnitud, por eso estas reglas son puestas bajo el punto de que producen un number (un valor double), mientras que las demás quedan definidas que generan un vector.

Pruebas.

Código.

Vector_cal.y

```
%{
    #include<stdio.h>
    #include<stdlib.h>
    #include<math.h>
    #include <ctype.h>
    #include"vector_cal.c"

    int yyerror(char *s);
    int yylex();
%}

%union{
    double val;
    Vector *vector;
}

%token<val> NUMBER

%type<vector> exp
%type<vector> vector
%type<val> number

%left '+' '-'
%left '*'
%left 'x' '.'

%%
    inputString:
      | inputString list;
      ;

    list: '\n'
      | exp     '\n'  {imprimeVector($1);}
      | number  '\n'  {printf("%lf\n", $1);}
      ;

    exp: vector
      | exp '+' exp     {$$ = sumaVector($1, $3);}
      | exp '-' exp     {$$ = restaVector($1, $3);}
      | exp '*' NUMBER     {$$ = escalarVector($3, $1);}
      | NUMBER '*' exp  {$$ = escalarVector($1, $3);}
      | exp 'x' exp     {$$ = productoCruz($1, $3);}
    ;

    number: NUMBER
      | vector '.' vector {$$ = productoPunto($1, $3);}
      | '|' vector '|' {$$ = vectorMagnitud($2);}
      ;

    vector : '[' NUMBER NUMBER NUMBER ']' {Vector *v = creaVector(3);
                                           v -> vec[0] = $2;
                                           v -> vec[1] = $3;
                                           v -> vec[2] = $4;
                                           $$ = v;}
      ;
```

```
%%

void main(){
  yyparse();
}

int yylex (){
    int c;
    while ((c = getchar ()) == ' ' || c == '\t')
        ;
    if (c == EOF)
            return 0;
  if(isdigit(c)){
      ungetc(c, stdin);
      scanf("%lf", &yylval.val);
      return NUMBER;
  }
  return c;
}

int yyerror(char *s){
  printf("%s\n", s);
  return 0;
}
```

Conclusiones.

Acerca de esta práctica el principal problema vino dado al no estar tan familiarizado con el lenguaje, pero después de varios intentos logre comprender el cómo adaptar las producciones al lenguaje YACC, de ahí todo fue sencillo ya que solo tuve que definir como se crean y que producen cada regla.