Instituto Politécnico Nacional

Escuela Superior de Cómputo

Departamento de Ciencias e Ingeniería

de la Computación

Compiladores

Práctica 1: Yacc Básico

Opción: Calculadora de vectores

Grupo: 5CM2

Alumno:

Sandoval Hernández Eduardo

Profesor:

Tecla Parra Roberto

Fecha de entrega:

18 de marzo de 2022

Introducción

Yet Another Compiler Compiler (YACC) es un generador de analizadores. A partir de un archivo fuente en yacc se puede generar un archivo fuente en C el cual contendrá el analizador sintáctico. Este analizador requerirá de un analizador léxico externo para funcionar, esto debido a que el archivo fuente en C que genera yacc contiene llamadas a la función yylex() la cual debe estar definida y debe ser capaz de devolver el tipo de lexema encontrado. Adicionalmente es necesario incluir una función yyerror() la cual será invocada cuando el analizador sintáctico encuentre un símbolo que no encaje con la gramática [1].

El presente trabajo tiene la finalidad de reportar el desarrollo de la práctica 1 con temática relacionada al uso de Yacc en un nivel básico, la opción elegida fue la calculadora de vectores la cual considerando que se tiene el código para obtener el producto punto, producto cruz, producto escalar, suma, resta y la obtención de la magnitud, se realiza la especificación en Yacc que permita evaluar expresiones que involucren operaciones con vectores. Para esta implementación se considerarán operaciones con vectores en 2, 3, 4 y 5 dimensiones, cabe aclarar que la operación de producto cruz no será posible realizarla con vectores en más de 3 dimensiones y por tanto el resultado será un vector nulo.

Desarrollo

Para la implementación se tienen previamente dos archivos vcalc.h y vcalc.c, en los cuales se encuentran los prototipos y la implementación de las funciones necesarias para realizar las operaciones con vectores, estas son: struct vector, que define el cómo se estructura un vector con sus componentes; creaVector, función encargada de crear un vector en n dimensiones; imprimeVector, función encargada de imprimir en pantalla los valores de las componentes del vector que recibe como parámetro; copiaVector, función encargada de realizar una copia de un vector recibido como parámetro; sumaVector, función que suma dos vectores y retorna el vector resultante; restaVector, función que resta dos vectores y retorna el vector resultante; escalarVector, función que realiza la multiplicación de un vector por un escalar y retorna el vector resultante; productoCruz, función que realiza el producto cruz de dos vectores y retorna el vector resultante; productoPunto, función que realiza el producto punto entre dos vectores y retorna el escalar resultante; vectorMagnitud, función que retorna la magnitud de un vector.

Para la implementación en Yacc se creó el archivo vcalc.y donde primero de definen las librerías que se utilizarán incluyendo los dos archivos anteriores para las operaciones con vectores, además de los prototipos para las funciones yylex y yyerror.

Posteriormente se definió el tipo de dato para la pila de Yacc como se muestra en el siguiente fragmento de código:

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

Esto debido a que en la pila se almacenarán los vectores y los valores de sus componentes.

Como símbolos terminales se definieron dos, uno para números positivos y otro para números negativos:

```
%token<val>     NUMBER      //  Terminal para números positivos
%token<val>     nNUMBER     //  Terminal para números negativos
```

En cuanto a los símbolos no terminales se definieron 4, uno para expresiones, otro para vectores, y dos como auxiliares para números:

```
%type<vector>   exp         //  No Terminal para expresiones
%type<vector>   vector      //  No Terminal para vectores
%type<val>      NTnumber    //  No Terminal auxiliar para números
%type<val>      auxNumber   //  Segundo No Terminal auxiliar para números
```

La jerarquía se definió de la siguiente manera:

```
%left '+' '-'               // Suma y resta
%left '*'                   // Producto de vector por escalar
%left 'x' '.'               // Producto cruz y producto punto
```

En cuanto a la gramática se tienen las siguientes producciones.

Una producción que permite el ingreso continuo de operaciones, es decir, se podrán procesar varias operaciones una por una conforme se va ingresando a la consola:

```
/*
    entrada => *vacio*
    entrada => entrada list
*/
    entrada :
            | entrada list;     // Esto funciona para que pueda realizar operación tras operación
            ;
```

La siguiente producción sirve como base para las siguientes producciones, ya que permite de manera general obtener el resultado de la expresión, ya sea un vector o un escalar:

```
/*
    list => '\n'            // Salto de linea
    list => exp '\n'
    list => NTnumber '\n'
*/
    list: '\n'
        | exp '\n'          {imprimeVector($1);}
        | NTnumber '\n'     {printf("\tResultado: %lf\n",$1);}
        ;
```

La siguiente producción permite evaluar las expresiones relacionadas a vectores simples, suma, resta, producto escalar y producto cruz de vectores:

```
/*
    exp => vector
    exp => exp '+' exp
    exp => exp '-' exp
    exp => exp '*' auxNumber
    exp => auxNumber * exp
    exp => exp 'x' exp
*/
    exp : vector
        | exp '+' exp           {$$ = sumaVector($1, $3);}      //  Suma de vectores
        | exp '-' exp           {$$ = restaVector($1, $3);}     //  Resta de vectores
        | exp '*' auxNumber     {$$ = escalarVector($3, $1);}   //  Producto escalar por la derecha
        | auxNumber '*' exp     {$$ = escalarVector($1, $3);}   //  Producto escalar por la izquierda
        | exp 'x' exp           {$$ = productoCruz($1, $3);}    //  Producto cruz de vectores
        ;
```

Las siguientes producciones funcionan para obtener el resultado del producto punto o magnitud de un vector y el manejo de números positivos y negativos:

```
/*
    NTnumber => auxNumber
    NTnumber => vector '.' vector
    NTnumber => '|' vector '|'
*/
    NTnumber: auxNumber                                         //  Numero negativo o positivo
            | vector '.' vector     {$$ = productoPunto($1, $3);}   //  Resultado del producto punto
            | '|' vector '|'        {$$ = vectorMagnitud($2);}      //  Resultado de la magnitud
            ;

/*
    auxNumber => NUMBER
    auxNumber => nNUMBER
*/
    auxNumber   : NUMBER            //  Número positivo
                | nNUMBER           //  Número negativo
                ;
```

La última producción es la encargada de procesar los vectores desde 2 hasta 5 dimensiones, apoyándose en la producción anterior y como acción en C la creación de los vectores:

```
/*
    vector => auxNumber auxNumber
    vector => auxNumber auxNumber auxNumber
    vector => auxNumber auxNumber auxNumber auxNumber auxNumber

*/
    vector  : '[' auxNumber auxNumber ']'   {   //  Crea vector de 2 dimensiones
                                                Vector *v = creaVector(2);
                                                v -> vec[0] = $2;
                                                v -> vec[1] = $3;
                                                $$ = v;
                                            }
            | '[' auxNumber auxNumber auxNumber ']' {   //  Crea vector de 3 dimensiones
                                                        Vector *v = creaVector(3);
                                                        v -> vec[0] = $2;
                                                        v -> vec[1] = $3;
                                                        v -> vec[2] = $4;
                                                        $$ = v;
                                                    }
            |  '[' auxNumber auxNumber auxNumber auxNumber ']'  {   //  Crea vector de 4 dimensiones
                                                                    Vector *v = creaVector(4);
                                                                    v -> vec[0] = $2;
                                                                    v -> vec[1] = $3;
                                                                    v -> vec[2] = $4;
                                                                    v -> vec[3] = $5;
                                                                    $$ = v;
                                                                }
            |  '[' auxNumber auxNumber auxNumber auxNumber auxNumber ']'    {   //  Crea vector de 5 dimensiones
                                                                                Vector *v = creaVector(5);
                                                                                v -> vec[0] = $2;
                                                                                v -> vec[1] = $3;
                                                                                v -> vec[2] = $4;
                                                                                v -> vec[3] = $5;
                                                                                v -> vec[4] = $6;
                                                                                $$ = v;
                                                                            }
            ;
```

Finalmente, para el código de soporte, la función main se limita a hacer la llamada a la función yyparse encargada del analizador sintáctico mientras la función yyerror se limita a imprimir en pantalla el mensaje de error predeterminado por Yacc. La función yylex encargada del analizador léxico se encargará de dividir la cadena de entrada en tokens, esta función se encarga de procesar los espacios, tabulaciones, corchetes para los vectores, operadores y números tanto positivos como negativos, para esto último, en caso de que algún componente del vector sea un número negativo este deberá ser escrito con un símbolo ‘n’ del lado izquierdo con la finalidad de que no exista confusión con el símbolo ‘-‘ que funge como operador para la resta de vectores, por ejemplo: si se desea escribir el vector [-2 4 2], se deberá escribir como [n2 4 2].

Para la compilación y ejecución del programa se ingresan los siguientes comandos en la consola:

```
eduardo@lalo-VM:~/Documents/Compiladores/vectorMine$ yacc -d vcalc.y
eduardo@lalo-VM:~/Documents/Compiladores/vectorMine$ gcc y.tab.c -lm
eduardo@lalo-VM:~/Documents/Compiladores/vectorMine$ ./a.out
```

Para confirmar el funcionamiento del programa se probaron todas las operaciones con vectores en las diversas dimensiones:

```
eduardo@lalo-VM:~/Documents/Compiladores/vectorMine$ yacc -d vcalc.y
eduardo@lalo-VM:~/Documents/Compiladores/vectorMine$ gcc y.tab.c -lm
eduardo@lalo-VM:~/Documents/Compiladores/vectorMine$ ./a.out
[1 2 3] + [2 4 6]
        Resultado: [ 3.000000 6.000000 9.000000 ]
[1 2 3 4 5] + [2 4 6 8 10]
        Resultado: [ 3.000000 6.000000 9.000000 12.000000 15.000000 ]
[1 2 3] - [2 n4 6]
        Resultado: [ -1.000000 6.000000 -3.000000 ]
[2 3 3] . [2 11 4]
        Resultado: 49.000000
[2 3 3] x [2 11 4]
        Resultado: [ -21.000000 -2.000000 16.000000 ]
[2 3 6] * n2
        Resultado: [ -4.000000 -6.000000 -12.000000 ]
|[43 22 50]|
        Resultado: 69.519781
[2 3 4 6] - [2 n7 n2 0]
        Resultado: [ 0.000000 10.000000 6.000000 6.000000 ]
[2 4] x [9 80]
        Resultado: [ 160.000000 -36.000000 ]
```

Como muestra la captura anterior, el programa es capaz de sumar y restar vectores desde 3 hasta 5 dimensiones y hacer los productos en dimensiones de 2 a 3.

Conclusión

Esta primera práctica ha servido para tener un primer acercamiento al funcionamiento de Yacc. Los códigos proporcionados previamente han facilitado el desarrollo de esta, pues solo ha sido necesario complementarlos con la especificación en Yacc. Al ser la primera práctica me ha resultado un poco más complicada de lo que creía, sin embargo, el resultado obtenido ha sido satisfactorio pues se tiene un programa funcional y simple con posibilidad de mejora para su uso en implementaciones en las prácticas siguientes.

Referencias

[1] F. Simmross Wattenberg (s.f.). “El generador de analizadores sintácticos yacc”. [Internet]. Disponible en https://www.infor.uva.es/~mluisa/talf/docs/labo/L8.pdf