Compiladores

- Práctica 01: calculadora de vectores -

Grupo 3CM7

Vargas Romero Erick Efraín
Prof. Tecla Parra Roberto

Instituto Politécnico Nacional
Escuela Superior de Cómputo
Juan de Dios Bátiz, nueva industrial Vallejo
07738 ciudad de México

Chapter 1

Práctica 01

1.1 Calculadora de vectores

1.1.1 Descripción

Esta primera práctica hace uso de Yacc para la creación de una calculadora de vectores. Esta calculadora permite realizar:

1. Suma de vectores
2. Resta de vectores
3. Multiplicación por un escalar
4. Producto punto
5. Producto cruz
6. Magnitud de un vector

1.1.2 Ejemplos

A continuación muestro una captura de pantalla, la cual muestra la compilación del código en yacc, y también la compilación del código que es generado en c y finalmente la ejecución del programa.

```
[erick@erick-pc Práctica 01]$ yacc -d vector_cal.y
[erick@erick-pc Práctica 01]$ gcc y.tab.c -lm
[erick@erick-pc Práctica 01]$ ./a.out
[1 1 1] + [1 1 1]
[ 2.000000 2.000000 2.000000 ]
[1 1 1] - [1 1 1]
[ 0.000000 0.000000 0.000000 ]
[1 1 1] . [1 1 1]
3.000000
[1 2 3] x [4 5 6]
[ -3.000000 6.000000 -3.000000 ]
|[1 2 3]|
3.741657
5*[1 2 3]
[ 5.000000 10.000000 15.000000 ]
[1 2 3]*5
[ 5.000000 10.000000 15.000000 ]
```

Como es posible observar, esta calculadora realiza las operaciones que fueron descritas en la sub-sección anterior. Hay que hacer hincapié en que es necesario colocar los símbolos "[" y "]" para indicar que hay un vector.

1.1.3 Código

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();
%}
//óDefinicin de tipos de dato de la pila de yacc
%union{
    double val;
    Vector *vector;
}

/**óCreacin de ísmbolos terminales y no terminales**/
%token<val>       NUMBER      //íSmbolo terminal

%type<vector>    exp         //S+imbolo no terminal
%type<vector>    vector     //íSmbolo no terminal
%type<val>          number       //íSmbolo no terminal
```

```
23
24 /**íJerarqua de operadores**/
25 //Suma y resta de vectores
26 %left '+' '-'
27 //Escalar por un vector
28 %left '*'
29 //Producto cruz y producto punto
30 %left 'x' '.'
31
32 /**áGramtica**/
33 %%
34   /**
35     * inputString -> inputString list
36   **/
37     inputString:
38       | inputString list;
39       ;
40
41   /**
42     * list -> '\n'          |
43     * list -> exp '\n'      |
44     * list -> number '\n'   |
45   **/
46     list: '\n'
47       | exp      '\n'  {imprimeVector($1);}
48       | number   '\n'  {printf("%lf\n", $1);}
49       ;
50
51     /**
52       * exp -> vector          |
53       * exp -> exp '+' exp     |
54       * exp -> exp '-' exp     |
55       * exp -> exp '*' NUMBER  |
56       * exp -> NUMBER * exp    |
57       * exp -> exp 'x' exp     |
58     **/
59     exp: vector
60     //Suma de vectores
61       | exp '+' exp      {$$ = sumaVector($1, $3);}
62     //Resta de vectores
63       | exp '-' exp      {$$ = restaVector($1, $3);}
64     //óMultiplicacin por un escalar caso 1
65       | exp '*' NUMBER     {$$ = escalarVector($3, $1);}
66     //óMultiplicacin por un escalar caso 2
67       | NUMBER '*' exp   {$$ = escalarVector($1, $3);}
68     //Producto cruz
69       | exp 'x' exp      {$$ = productoCruz($1, $3);}
70     ;
71     /**
72       * number -> NUMBER
73
74     **/
75     number: NUMBER
```

```
76        | vector '.' vector {$$ = productoPunto($1, $3);}
77        | '|' vector '|' {$$ = vectorMagnitud($2);}
78        ;
79
80      /**
81        * vector -> NUMBER NUMBER NUMBER
82      **/
83      vector : '[' NUMBER NUMBER NUMBER ']' {Vector *v = creaVector(3);
84                                             v -> vec[0] = $2;
85                                             v -> vec[1] = $3;
86                                             v -> vec[2] = $4;
87                                             $$ = v;}
88        ;
89 %%
90
91 /*************************************************************
92 * ó                        Cdigo en C                        *
93 *************************************************************/
94
95 void main(){
96   yyparse();
97 }
98
99 int yylex (){
100     int c;
101     while ((c = getchar ()) == ' ' || c == '\t')
102       ;
103   if (c == EOF)
104         return 0;
105   if(isdigit(c)){
106       ungetc(c, stdin);
107       scanf("%lf", &yylval.val);
108       return NUMBER;
109   }
110   return c;
111 }
112
113 int yyerror(char *s){
114   printf("%s\n", s);
115   return 0;
116 }
```

Código vector_cal.c

```
1 #include <stdio.h>
2 #include "vector_cal.h"
3 #include <ctype.h>
4 #include <stdlib.h>
5 #include <math.h>
6
7 /**
8  * Esta ófuncin crea un vector de n dimensiones.
9  * Pero para este caso nos conformaremos con
10 * crear vectores de a lo mucho 3 dimensiones.
```

```
11  * @param: int n (ódimensin del vector)
12  * @return: vector*
13  * */
14 Vector *creaVector(int n){
15     Vector *vec;
16     int i;
17     vec=(Vector *)malloc(sizeof(Vector));
18     vec->n = n;
19     vec->vec = (double *)malloc(sizeof(double)*n);
20     return vec;
21 }
22
23 /**
24  * Esta ófuncin imprime los elementos que
25  * contiene el vector que es pasado como áparmetro
26  * @param: * vector
27  * @return: nada
28  * */
29 void imprimeVector(Vector *v){
30     int i;
31     printf("[ ");
32     for(i=0; i< v->n; i++)
33         printf("%f ", v->vec[i]);
34     printf("]\n");
35 }
36
37 /**
38  * Esta ófuncin copia el contenido de un vector
39  * @param: *vector
40  * @return: *vector
41  * */
42 Vector *copiaVector(Vector *v){
43     int i;
44     Vector *copy=creaVector(v->n);
45     for(i = 0; i< v->n; i++)
46         copy->vec[i]=v->vec[i];
47     return copy;
48 }
49
50 /**
51  * Esta ófuncin suma dos vectores los cuales
52  * recibe la ófuncin y retorna el nuevo
53  * vector.
54  * @param: *vector, *vector
55  * @return: *vector
56  * */
57 Vector *sumaVector(Vector *a, Vector *b){
58 Vector *c;
59 int i;
60     c=creaVector(a->n);
61     for(i=0; i< a->n;i++)
62         c->vec[i]=a->vec[i]+b->vec[i];
63     return c;
```

```
64 }
65
66 /**
67  * Esta ófuncin resta dos vectores los cuales
68  * recibe la ófuncin y retorna el nuevo
69  * vector,
70  * @param: *vector, *vector
71  * @return: *vector
72  * */
73 Vector *restaVector(Vector *a, Vector *b){
74 Vector *c;
75 int i;
76    c=creaVector(a->n);
77    for(i=0; i< a->n;i++)
78       c->vec[i]=a->vec[i]-b->vec[i];
79    return c;
80 }
81
82 /**
83  * Esta ófuncin multiplica un escalar por
84  * un vector.
85  * @param: double, *vector
86  * @return: vector
87  * */
88 Vector  *escalarVector(double c, Vector *v){
89
90     Vector *r_vector = creaVector(v -> n);
91     int i;
92     /*Iteramos mientras el índice sea menor que el ñtamao
93     del vector
94         */
95     for(i = 0; i < v -> n; i++){
96         /*El vector resultante en su elemento i
97          es igual al producto del escalar por el
98          vector recibido en el argumento en suó
99          posicin i
100             */
101         r_vector -> vec[i] = c * v->vec[i];
102     }
103   return r_vector;
104 }
105
106 /**
107  * Esta ófuncin realiza el producto cruz de dos vectores
108  * los cuales recibe la ófuncin. áAdems
109  * retorna el nuevo vector calculado.
110  * @param: *vector, *vector
111  * @return: *vector
112  * */
113 Vector *productoCruz(Vector *a, Vector *b){
114   Vector *r;
```

```
113   r = creaVector(a -> n);
114
115   //Si el vector tiene una ódimensin de dos
116   if(a-> n == 2){
117         r -> vec[0] = a -> vec[0] * b ->vec[1];
118         r -> vec[1] -= a -> vec[1] * b -> vec[0];
119     }
120     //Si el vector tiene una ódimensin de tres
121     else if(a -> n == 3){
122         r -> vec[0] = a -> vec[1] * b -> vec[2]
123             - a -> vec[2] * b -> vec[1];
124
125         r -> vec[1] = a -> vec[2] * b -> vec[0]
126             - a -> vec[0] * b -> vec[2];
127
128         r -> vec[2] = a -> vec[0] * b -> vec[1]
129             - a->vec[1] * b -> vec[0];
130     }
131     return r;
132 }
133
134 /**
135  * Esta ófuncin realiza el producto punto entre dos vectores.
136  * Como sabemos el producto punto entre dos vectores
137  * nos da como resultado un escalar. En este caso double
138  * @param: *vector, *vector
139  * @return: double
140  * */
141 double productoPunto(Vector *a, Vector *b){
142     double resultado = 0.0f;
143     int i;
144     for(i = 0; i < a->n; i++){
145         //Acumulamos el resultado del producto de cada componente
146         resultado += ( a -> vec[i] * b->vec[i] );
147     }
148     return resultado;
149 }
150
151 /**
152  * Esta ófuncin calcula la magnitud de un vector.
153  * Recordemos que la magnitud de un vector es
154  * igual ha el producto punto entre
155  * */
156 double vectorMagnitud(Vector *a){
157     double resultado = 0.0f;
158     int i;
159     for(i = 0; i < a->n; i++){
160         resultado += ( a -> vec[i] * a -> vec[i] );
161     }
162     resultado = sqrt(resultado);
163     return resultado;
164 }
```

La estructura Vector y las definiciones de nuestras funciones se encuentran en el archivo vector_cal.h

```
1
2  struct vector {
3    char name;
4    int n;
5    double *vec;
6  };
7  typedef struct vector Vector;
8  //óCreacin de un vector
9  Vector *creaVector(int n);
10 //óImpresin de un vector
11 void imprimeVector(Vector *a);
12 //Copiado de vectores
13 Vector *copiaVector(Vector *a);
14 //Suma de vectores
15 Vector *sumaVector(Vector *a, Vector *b);
16 //Resta de vectores
17 Vector *restaVector(Vector *a, Vector *b);
18 //Multiplica un vector por un escalar
19 Vector *escalarVector(double c, Vector *v);
20 //Producto cruz entre dos vectores
21 Vector *productoCruz(Vector *a, Vector *b);
22 //Producto punto entre vectores
23 double productoPunto(Vector *a, Vector *b);
24 //áClculo de la magnitud de un vector
25 double vectorMagnitud(Vector *a);
```